GraphBolt: Dependency-Driven Synchronous Processing of Streaming Graphs

Mugilan Mariappan and Keval Vora
Simon Fraser University

EuroSys’19 – Dresden, Germany
March 27, 2019
Graph Processing
Dynamic Graph Processing

Real-time Processing
• Low Latency

Real-time **Batch** Processing
• High Throughput

Alipay payments unit of Chinese retailer Alibaba [...] has 120 billion nodes and over 1 trillion relationships [...]; this graph has **2 billion updates each day** and was running at **250,000 transactions per second** on Singles Day [...]

Streaming Graph Processing

Incremental Processing

• Adjust results **incrementally**
• Reuse work that has already been done

**Tag Propagation**
upon mutation

Over 75% values get thrown out

KickStarter [ASPLOS’17]

<table>
<thead>
<tr>
<th>Tornado [SIGMOD’16]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphIn [EuroPar’16]</td>
</tr>
<tr>
<td>KineoGraph [EuroSys’12]</td>
</tr>
</tbody>
</table>

Streaming Graph Processing

Incremental Processing

- Adjust results **incrementally**
- Reuse work that has already been done

|----------------------|-----------------------|-------------------------|-------------------------|

Maintain **Value Dependences**

Incrementally refine results

*Less than 0.0005%* values thrown out

*Kickstarter: Fast and Accurate Computations on Streaming Graph via Trimmed Approximations. Vora et al., ASPLOS 2017.*
Streaming Graph Processing

Incremental Processing
• Adjust results **incrementally**
• Reuse work that has already been done

Monotonic Graph Algorithms

Maintain **Value Dependences**
Incrementally refine results

Less than 0.0005% values thrown out

Streaming Graph Processing

• Belief Propagation
• Co-Training Expectation Maximization
• Collaborative Filtering
• Label Propagation
• Triangle Counting
• …

![Graph showing incorrect vertices and updates]

- Error > 10%
- Incorrect Vertices vs. Updates
- Updates: 1, 2, 3, 4, 5
Streaming Graph Processing

Incorrect Vertices

Updates

Error > 10%
Streaming Graph Processing

G mutates at k

I \rightarrow R_G^k \rightarrow R_G

Incorrect Vertices

Updates

Error > 10%

Updates
Streaming Graph Processing

G mutates at k

I → $R^k_G$ → $R^k_G$ → $R_{G,M}$

$R^k_? \neq R^k_{G,M}$

Error > 10%

Incorrect Vertices

Updates

0 1 2 3 4 5

Incorrect Vertices

Updates

0 1 2 3 4 5
Streaming Graph Processing

G mutates at k

I → R^k_G → R_G

Z^s(R^k_G)

R^{k+1}_? → R? ≠ R^k_{GM}

R^k_{GM} → R^k_{GM}

G mutates at k

Incorrect Vertices

Error > 10%

Updates

Incorrect Vertices

Updates
GraphBolt

- **Dependency-Driven Incremental Processing of Streaming Graphs**
- Guarantee **Bulk Synchronous Parallel** Semantics
- Lightweight dependence tracking
- Dependency-aware value refinement upon graph mutation
Bulk Synchronous Processing (BSP)

∀(u, v) ∈ E, \( u^t \rightarrow v^{t+1} \)

Streaming Graph
Upon Edge Addition

\[ \forall (u, v) \in E, \ u^t \mapsto v^{t+1} \]

Streaming Graph
Upon Edge Addition

∀(u, v) ∈ E, \( u^t \rightarrow v^{t+1} \)
Upon Edge Addition

\[ \forall (u, v) \in E, \ u^t \mapsto v^{t+1} \]
Upon Edge Addition

∀(u, v) ∈ E, \( u^t \rightarrow v^{t+1} \)
Upon Edge Addition

\[ \forall (u, v) \in E, \ u^t \mapsto v^{t+1} \]

**Vertices**

**Iteration**

**Streaming Graph**

**Ideal Scenario**
Upon Edge Addition

\[ \forall (u, v) \in E, \quad u^t \mapsto v^{t+1} \]

Streaming Graph

Ideal Scenario

Vertices
Upon Edge Addition

\[ \forall (u, v) \in E, \ u^t \rightarrow v^{t+1} \]
Upon Edge Addition

\[ \forall (u, v) \in E, \quad u^t \mapsto v^{t+1} \]

Vertices

Streaming Graph

Ideal Scenario

Iteration

Vertices
Upon Edge Addition

\[ \forall (u, v) \in E, \ u^t \mapsto v^{t+1} \]

Streaming Graph

Ideal Scenario
Upon Edge Deletion

\[ \forall (u, v) \in E, \ u^t \rightarrow v^{t+1} \]

**Vertices**

**Streaming Graph**

**Ideal Scenario**
GraphBolt: Correcting Dependencies

\[
\forall (u, v) \in E, \ u_t \rightarrow v^{t+1}
\]
GraphBolt: Correcting Dependencies

\[
\forall (u, v) \in E, \ u^t \mapsto v^{t+1}
\]

Vertices

Ideal Scenario

Streaming Graph

Iteration

GraphBolt: Correcting Dependencies

\[
\forall (u, v) \in E, \ u^t \mapsto v^{t+1}
\]

Vertices

Ideal Scenario

Streaming Graph

Iteration
GraphBolt: Correcting Dependencies

\[ \forall (u, v) \in E, \ u^t \rightarrow v^{t+1} \]

Vertices

Iteration

Streaming Graph

Ideal Scenario
GraphBolt: Correcting Dependencies

Ideal Scenario

∀(u, v) ∈ E, u⁰ → vᵗ⁺¹

Vertices

Iteration

v
u
y

x

k-2
k-1
k

Streaming Graph

k-2
k-1
k

Ideal Scenario

Vertices
GraphBolt: Correcting Dependencies

Ideal Scenario

\[ \forall (u, v) \in E, \ u^t \rightarrow v^{t+1} \]

Vertices

Streaming Graph

Vertices

Ideal Scenario
GraphBolt: Correcting Dependencies

Ideal Scenario

\[ \forall (u, v) \in E, \ u^t \mapsto v^{t+1} \]

Iteration

Vertices

Vertices

GraphBolt: Correcting Dependencies

Streaming Graph

Ideal Scenario
GraphBolt: Dependency Tracking

\[
V = \bigcap (\bigotimes (c_{i-1}(u)))
\]

\[
c_i(v) = \bigcap (\bigotimes (c_{i-1}(u))) \quad \forall e=(u,v) \in E
\]

\[
g_i(v) = \bigoplus (c_{i-1}(u)) \quad \forall e=(u,v) \in E
\]
GraphBolt: Dependency Tracking

- Dependency relations translate across aggregation points
- Structure of dependencies inferred from input graphs

\[
V = \bigcap (\bigoplus (c_{i-1}(u)))
\]

\[
c_i(v) = \bigcap \left( \bigoplus (c_{i-1}(u)) \right)
\]

\[
g_i(v) = \bigoplus (c_{i-1}(u))
\]
GraphBolt: Incremental Refinement

\[ g_i^T(v) = g_i(v) + ? \]
GraphBolt: Incremental Refinement

- **Vertices**
- **Refinement**
  - **Direct changes**
  - **Transitive changes**
    - **Edge additions**
      \[ \bigcup_{e=(u,v)\in E_a} (c_{i-1}(u)) \]
    - **Edge deletions**
      \[ \bigcup_{e=(u,v)\in E_d} (c_{i-1}(u)) \]

$$g_i^T(v) = g_i(v) \bigcup_{e=(u,v)\in E_a} (c_{i-1}(u)) \bigcap_{e=(u,v)\in E_d} (c_{i-1}(u)) + ?$$
GraphBolt: Incremental Refinement

We focus our discussion on incremental aggregation operators that act as sub-operations to perform certain vertices altogether, it would require backpropagation during recomputation to be incrementally computed for our dependency graph. For example in Alternating Least Squares operate on vectors or multi-valued variables that interact with elements of other complex aggregation. For instance in Alternating Least Squares cannot be directly formulated. The right-hand side of the above equation is referred to as the contribution of vertex 2’s outgoing neighbors 0 and 1 (dotted edges), which means, we dynamically identify the aggregation values in the previous iteration before which the graph mutates, we aim to structure of graphs in the latest iteration. At each iteration, we ask two questions that help us transform first, what values corresponding to end points of edges to be incrementally computed for our dependency graph. Let $G$ be a graph defined as the process of refinement. To do so, we start with aggregation values in $T$. Figure 5 shows how the refinement process are far lesser than that involved under synchronous semantics. To effect of dependency-driven incremental processing.

$g^T_i(v) = g_i(v) \bigoplus (c_{i-1}(u)) \bigcap (c_{i-1}(u)) \bigtriangleup (c^T_{i-1}(u))$

\[
\forall e = (u, v) \in E_a \quad \forall e = (u, v) \in E_d \\
\forall e = (u, v) \in E^T s.t. c_{i-1}(u) \neq c^T_{i-1}(u)
\]
Refinement: Transitive Changes

Formally, at the end of iteration $i$, i.e., $i=1$, 2, ..., $k$, $\forall u \in V$: the dependency graph in terms of aggregation values at the end of mutation by incrementally correcting the aggregated values at vertices, we can track these aggregated values. It is interesting to note that the structure of dependencies among intermediate values based on Eq. 1.

$g_i(v) = \bigoplus_{\forall e=(u,v) \in E} (c_{i-1}(u))$
Refinement: Transitive Changes

Vertex aggregation

Incremental refinement

Aggregation

Complex

- Retract : Old value
- Propagate : New value

Belief Propagation

\[
\forall s \in S : \prod_{\forall e=(u,v)\in E} \left( \sum_{\forall s'\in S} \phi(u, s') \times \psi(u, v, s', s) \times \frac{c(u, s')}{c(v, s')} \right)
\]
Refinement: Transitive Changes

Vertex aggregation

Incremental refinement

Aggregation

Complex

- Retract: Old value
- Propagate: New value

Belief Propagation

\[ g_i(v) = \bigoplus_{\forall e=(u,v)\in E} (c_{i-1}(u)) \]

\[ \forall s \in S : \prod_{\forall e=(u,v)\in E} \left( \sum_{\forall s'\in S} \phi(u,s') \times \psi(u,v,s',s) \times \frac{c(u,s')}{c(v,s')} \right) \]
Refinement: Aggregation Types

Vertex aggregation

Incremental refinement

Aggregation

Complex | Simple

Propagate: Change in vertex value

- (■) - Transform
- (■)(■) - Combine

\[ g_i(v) = \bigoplus_{\forall e=(u,v) \in E} (c_{i-1}(u)) \]
Refinement: Aggregation Types

Vertex aggregation

Incremental refinement

Aggregation

Complex

Simple

Propagate: Change in vertex value

Vertex aggregation

Incremental refinement

Aggregation

Complex

Simple

Propagate: Change in vertex value

PageRank

Aggregation

Transform

Vertex Value
GraphBolt: Programming Model

```plaintext
function REPROPAGATE(e = (u, v), i)
  ATOMICADD(sum[v][i + 1], oldpr[u][i] / old_degree[u])
end function

function RETRACT(e = (u, v), i)
  ATOMICSUB(sum[v][i + 1], oldpr[u][i] / old_degree[u])
end function

function PROPAGATE(e = (u, v), i)
  ATOMICADD(sum[v][i + 1], newpr[u][i] / new_degree[u])
end function

function PAGERANK()
  for i ∈ [0...k] do
    V_updated = getSources(E_add ∪ E_delete)
    V_change = getTargets(E_add ∪ E_delete)
    for i ∈ [0...k] do
      E_update = {(u, v) : u ∈ V_updated} ∪ E_add
      EDGEMAP(E_update, RETRACT, i)
      EDGEMAP(E_update, PROPAGATE, i)
      V_dest = getTargets(E_update)
      V_change = V_change ∪ V_dest
      V_updated = VERTEXMAP(V_change, COMPUTE, i)
    end for
  end for
end function
```

Direct changes
GraphBolt: Programming Model - Complex aggregations

**Algorithm 1**

```
function REPROPAGATE(e = (u, v), i)
    ATOMICADD(&sum[v][i + 1], \frac{oldpr[u][i]}{old\_degree[u]})
end function
```

```
function RETRACT(e = (u, v), i)
    ATOMICSUB(&sum[v][i + 1], \frac{oldpr[u][i]}{old\_degree[u]})
end function
```

```
function PROPAGATE(e = (u, v), i)
    ATOMICADD(&sum[v][i + 1], \frac{newpr[u][i]}{new\_degree[u]})
end function
```

**Algorithm 2**

```
function PAGERANK()
    for i ∈ [0...k] do
        EDGEMAP(E_add, REPROPAGATE, i)
        EDGEMAP(E_delete, retract, i)
    end for
    V_updated = getSources(E_add ∪ E_delete)
    V_change = getTargets(E_add ∪ E_delete)
    for i ∈ [0...k] do
        E_update = \{(u, v) : u ∈ V_updated\}  // Transitive changes
        EDGEMAP(E_update, RETRACT, i)
        EDGEMAP(E_update, PROPAGATE, i)
        V_dest = getTargets(E_update)
        V_change = V_change ∪ V_dest
        V_updated = VERTEXMAP(V_change, COMPUTE, i)
    end for
end function
```
GraphBolt: Programming Model - Simple aggregations

**Algorithm 1**

```plaintext
function REPROPAGATE(e = (u, v), i)
    ATOMICADD(&sum[v][i + 1], oldpr[u][i] / old_degree[u])
end function

function RETRACT(e = (u, v), i)
    ATOMICSUB(&sum[v][i + 1], oldpr[u][i] / old_degree[u])
end function

function PROPAGATEDelta(e = (u, v), i)
    ATOMICADD(&sum[v][i + 1], newpr[u][i] / new_degree[u] - oldpr[u][i] / old_degree[u])
end function
```

**Algorithm 2**

```plaintext
function PAGE-RANK( )
    for i ∈ [0...k] do
        EDGE-MAP(E_add, REPROPAGATE, i)
        EDGE-MAP(E_delete, retract, i)
    end for
    V_updated = GET SOURCES(E_add ∪ E_delete)
    V_change = GET TARGETS(E_add ∪ E_delete)
    for i ∈ [0...k] do
        E_update = {(u, v) : u ∈ V_updated}  Transitive changes
        EDGE-MAP(E_update, PROPAGATE-Delta, i)
        V_dest = GET TARGETS(E_update)
        V_change = V_change ∪ V_dest
        V_updated = VERTEX-MAP(V_change, compute, i)
    end for
end function
```
GraphBolt Details …

- **Aggregation properties** & **non-decomposable** aggregations
- **Pruning dependencies** for light-weight tracking
  - Vertical pruning
  - Horizontal pruning
- **Hybrid incremental** execution
  - With & without dependency information
- Graph data structure & parallelization model
Experimental Setup

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Aggregation (⊕)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank (PR)</td>
<td>$\sum_{\forall e=(u,v) \in E} \frac{c(u)}{out _degree(u)}$</td>
</tr>
<tr>
<td>Belief Propagation (BP)</td>
<td>$\forall s \in S : \prod_{\forall e=(u,v)\in E} (\sum_{\forall s' \in S} \phi(u, s') \times \psi(u, v, s', s) \times c(u, s'))$</td>
</tr>
<tr>
<td>Label Propagation (LP)</td>
<td>$\forall f \in F : \sum_{\forall e=(u,v)\in E} c(u, f) \times weight(u, v)$</td>
</tr>
<tr>
<td>Co-Training Expectation Maximization (CoEM)</td>
<td>$\sum_{\forall e=(u,v)\in E} \frac{c(u) \times weight(u,v)}{\sum_{\forall e=(w,v)\in E} \frac{weight(w,v)}{}}$</td>
</tr>
<tr>
<td>Collaborative Filtering (CF)</td>
<td>$\langle \sum_{\forall e=(u,v)\in E} c_l(u) \cdot c_l(u)^{tr}, \sum_{\forall e=(u,v)\in E} c_l(u) \cdot weight(u,v) \rangle$</td>
</tr>
<tr>
<td>Triangle Counting (TC)</td>
<td>$\sum_{\forall e=(u,v)\in E}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graphs</th>
<th>Edges</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiki (WK) [47]</td>
<td>378M</td>
<td>12M</td>
</tr>
<tr>
<td>UKDomain (UK) [7]</td>
<td>1.0B</td>
<td>39.5M</td>
</tr>
<tr>
<td>Twitter (TW) [21]</td>
<td>1.5B</td>
<td>41.7M</td>
</tr>
<tr>
<td>TwitterMPI (TT) [8]</td>
<td>2.0B</td>
<td>52.6M</td>
</tr>
<tr>
<td>Friendster (FT) [14]</td>
<td>2.5B</td>
<td>68.3M</td>
</tr>
<tr>
<td>Yahoo (YH) [49]</td>
<td>6.6B</td>
<td>1.4B</td>
</tr>
</tbody>
</table>

Server: 32-core / 2 GHz / 231 GB
GraphBolt Performance

- **PR on TT**
- **CF on TT**
- **BP on TT**
- **COEM on FT**
- **LP on FT**
- **TC on FT**

Execution time (seconds):

- **GraphBolt**
- **GraphBolt-Reset**
- **Ligra**

(normalized computation time)
GraphBolt Performance

PR on TT (1.4x to 1.6x)

CF on TT (3.8x to 8.6x)

BP on TT (8.6x to 23.1x)

COEM on FT (28.5x to 32.3x)

LP on FT (5.9x to 24.7x)

TC on FT (279x to 722.5x)

GraphBolt-Reset

GraphBolt

Normalized computation time

GB-Reset

GraphBolt-Reset

GraphBolt
GraphBolt Performance

- PR on TT (1.4x to 1.6x)
- CF on TT (3.8x to 8.6x)
- COEM on FT (28.5x to 32.3x)
- LP on FT (5.9x to 24.7x)
- BP on TT (8.6x to 23.1x)
- TC on FT (279x to 722.5x)

GraphBolt-Reset
GraphBolt
100%
0.2%
3.6%
9.3%
GraphBolt Performance

- **PR on TT (1.4x to 1.6x)**
- **CF on TT (3.8x to 8.6x)**
- **BP on TT (8.6x to 23.1x)**
- **COEM on FT (28.5x to 32.3x)**
- **LP on FT (5.9x to 24.7x)**
- **TC on FT (279x to 722.5x)**

Normalized computation time for different GraphBolt and GraphBolt-Reset operations.
Detailed Experiments …

- Varying workload type
  - Mutations affecting majority of the values v/s affecting small values
- Varying graph mutation rate
  - Single edge update (reactiveness) v/s 1 million edge updates (throughput)
- Scaling to large graph (Yahoo) over large system (96 core / 748 GB)
GraphBolt v/s Differential Dataflow

Differential Dataflow vs. GraphBolt:

- **Execution Time (seconds)**
  - **Batch Size**: 1, 10, 100, 1K, 10K
  - **Execution Time**: 0, 125, 250, 375, 500, 625

**PR on TT**

- **Differential Dataflow**: 7.6x faster
- **GraphBolt**: 41.7x faster

Differential Dataflow. McSherry et al., CIDR 2013.
Summary

• Efficient **incremental processing** of **streaming graphs**
• Guarantee **Bulk Synchronous Parallel** semantics
• Lightweight **dependence tracking** at aggregation level
• **Dependency-aware value refinement** upon graph mutation
  • Programming model to support **incremental complex aggregation** types

Acknowledgements