X-Stream: Edge-centric Graph Processing using Streaming Partitions

A. Roy, I. Mihailovic, W. Zwaenepoel

Presented by: Samuil Stoychev
Graphs Processing

• Growing use in social networks, web rankings and others.
• Modern graphs can contain billions of edges.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiveJournal [9]</td>
<td>4.8M</td>
<td>69M</td>
</tr>
<tr>
<td>Twitter 2010 [31]</td>
<td>42M</td>
<td>1.5B</td>
</tr>
<tr>
<td>Yahoo web [8]</td>
<td>1.4B</td>
<td>6.6B</td>
</tr>
</tbody>
</table>

Source: “One Trillion Edges: Graph Processing at Facebook-Scale” (Ching et al., 2015)
Graph Processing Frameworks

- **Distributed** *(scale out)* frameworks:
  - Giraph
  - Pregel
  - Powergraph

- **Single-machine** *(scale up)* frameworks:
  - Graphchi
  - X-Stream
The Scatter-Gather Programming Model

- State is maintained in the vertices.
- User provides a **scatter** and a **gather** function.
- Scatter propagates updates to neighbours
- Gather accumulates updates from neighbours.
The Scatter-Gather Programming Model

- State is maintained in the vertices.
- User provides a \texttt{scatter} and a \texttt{gather} function.
- Scatter propagates updates to neighbours.
- Gather accumulates updates from neighbours.
The Scatter-Gather Programming Model

- State is maintained in the vertices.
- User provides a **scatter** and a **gather** function.
- Scatter propagates updates to neighbours.
- Gather accumulates updates from neighbours.
The Scatter-Gather Programming Model

- vertex_scatter(vertex v)
  - send updates over outgoing edges of v

- vertex_gather(vertex v)
  - apply updates from inbound edges of v

while not done
  - for all vertices v that need to scatter updates
    - vertex_scatter(v)
  - for all vertices v that have updates
    - vertex_gather(v)
The Scatter-Gather Programming Model

vertex_scatter(vertex v)
   send updates over outgoing edges of v

vertex_gather(vertex v)
   apply updates from inbound edges of v

while not done
   for all vertices v that need to scatter updates
      vertex_scatter(v)
   for all vertices v that have updates
      vertex_gather(v)
The Scatter-Gather Programming Model

- Simple but powerful interface.
- Sufficient to express a variety of algorithms.
- Used by Pregel and Powergraph.
Vertex-Centric Scatter-Gather (BFS)

Disk

(A,B)  (A,D)  (C,A)  (C,E)  (D,E)  (E,B)  (F,E)  (G,A)  (G,F)

Disk
Vertex-Centric Scatter-Gather (BFS)

(A,B)  (A,D)  (C,A)  (C,E)  (D,E)  (E,B)  (F,E)  (G,A)  (G,F)

(A,B)  (A,D)  (C,A)  (C,E)  (D,E)  (E,B)  (F,E)  (G,A)  (G,F)

Disk

Disk
Vertex-Centric Scatter-Gather (BFS)
Vertex-Centric Scatter-Gather (BFS)

Disk

(A,B)
(A,D)
(C,A)
(C,E)
(D,E)
(E,B)
(F,E)
(G,A)
(G,F)

Disk
Vertex-Centric Scatter-Gather (BFS)
Vertex-Centric Scatter-Gather (BFS)
Random vs Sequential Access

- Random memory access is slower than sequential memory access.
- Especially problematic for disk devices.
- Programs need to exploit locality to achieve efficient memory access.

Source: “Computer Systems: A Programmer’s Perspective” (Bryant and O’Hallaron)
Random vs Sequential Access

<table>
<thead>
<tr>
<th>Medium</th>
<th>Read (MB/s)</th>
<th>Write (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>Sequential</td>
</tr>
<tr>
<td>RAM (1 core)</td>
<td>567</td>
<td>2605</td>
</tr>
<tr>
<td>RAM (16 cores)</td>
<td>14198</td>
<td>25658</td>
</tr>
<tr>
<td>SSD</td>
<td>22.5</td>
<td>667.69</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>0.6</td>
<td>328</td>
</tr>
</tbody>
</table>

- Magnetic disk reads are 500+ times slower for random access.
- The gap in performance is bigger for slower media.
X-Stream

• Graph processing on a **single shared-memory machine**.
• Minimises random memory access through:
  • Edge-centric scatter-gather
  • Streaming partitions
• Supports both in-memory and out-of-core graphs.
Edge-Centric Scatter-Gather

\[
\text{edge\_scatter}(\text{edge } e) \\
\quad \text{send update over } e
\]

\[
\text{update\_gather}(\text{update } u) \\
\quad \text{apply update } u \text{ to } u.\text{destination}
\]

\[\text{while not done} \hspace{0.5cm} \begin{align*} \\
\quad & \text{for all edges } e \\
\quad & \quad \text{edge\_scatter}(e) \\
\quad & \text{for all updates } u \\
\quad & \quad \text{update\_gather}(u)
\end{align*}\]

- **Streaming** (or iterating over) edges instead of vertices.
Edge-Centric Scatter-Gather (BFS)
Edge-Centric Scatter-Gather (BFS)
Edge-Centric Scatter-Gather (BFS)
Edge-Centric Scatter-Gather (BFS)

PROBLEM: VERTEX MEMORY ACCESS STILL RANDOM
Edge-Centric Scatter-Gather (BFS)
Edge-Centric Scatter-Gather (BFS)

PROBLEM: VERTICES MIGHT NOT FIT IN MAIN MEMORY
Streaming Partitions

- Split the set of vertices into partitions such that every partition fits in memory.
- A partition also includes all edges whose source vertex is within that partition.
Streaming Partitions

PARTITION 1

(A,B)  
B  
(C,A)  
D  
(C,E)  
(D,E)  
(E,B)  
(F,E)  
(G,A)  
(G,F)

PARTITION 2

E  
F  
G  
(E,B)  
(F,E)  
(G,A)  
(G,F)

Main memory

Disk

Main memory

Disk

(A,D)  
(C,A)  
(C,E)  
(D,E)  
(E,B)  
(F,E)  
(G,A)  
(G,F)

Main memory

Disk

Main memory

Disk

A

B

C

D

E

F

G
Shuffle Phase

PARTITION 1

<table>
<thead>
<tr>
<th>A</th>
<th>(A,B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>(A,D)</td>
</tr>
<tr>
<td>C</td>
<td>(C,A)</td>
</tr>
<tr>
<td>D</td>
<td>(C,E)</td>
</tr>
<tr>
<td>E</td>
<td>(D,E)</td>
</tr>
</tbody>
</table>

Main memory | Disk

PARTITION 2

<table>
<thead>
<tr>
<th>E</th>
<th>(E,B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>(F,E)</td>
</tr>
<tr>
<td>G</td>
<td>(G,A)</td>
</tr>
<tr>
<td>(G,F)</td>
<td></td>
</tr>
</tbody>
</table>

Main memory | Disk

Main memory | Disk

PARTITION 1

<table>
<thead>
<tr>
<th>A</th>
<th>(A,B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>(A,D)</td>
</tr>
<tr>
<td>C</td>
<td>(C,A)</td>
</tr>
<tr>
<td>D</td>
<td>(C,E)</td>
</tr>
<tr>
<td>E</td>
<td>(D,E)</td>
</tr>
</tbody>
</table>

Main memory | Disk

PARTITION 2

<table>
<thead>
<tr>
<th>E</th>
<th>(E,B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>(F,E)</td>
</tr>
<tr>
<td>G</td>
<td>(G,A)</td>
</tr>
<tr>
<td>(G,F)</td>
<td></td>
</tr>
</tbody>
</table>

Main memory | Disk
Shuffle Phase

PARTITION 1
- A
- B
- C
- D
- Main memory
- Disk

U_OUT
- (A,B)
- (A,D)
- (C,A)
- (C,E)
- (D,E)

U_IN(1)
- (A,B)

PARTITION 2
- E
- F
- G
- Main memory
- Disk

U_IN(2)
- (D,E)

Main memory

Disk
- E
- F
- G
- (E,B)
- (F,E)
- (G,A)
- (G,F)
Generalising the Approach

• We showed interaction between:
  • **Disk** (*Slow Storage*)
  • **Main memory** (*Fast Storage*)

• But the same concept can be applied to
  • Main memory (*Slow Storage*)
  • Cache (*Fast Storage*)

• This allows to apply X-Stream for support both in-memory and out-of-core graphs specifying two engines:
  • **Out-of-core Streaming Engine**
  • **In-memory Streaming Engine**
Evaluation Setup

• 1U Server:
  • 64GB main memory
  • 200GB SSD
  • 3TB magnetic disks

• X-Stream evaluated over 10 popular algorithms.

• Both synthetic and real-world datasets.
# Algorithmic Performance

<table>
<thead>
<tr>
<th>memory</th>
<th>WCC</th>
<th>SCC</th>
<th>SSSP</th>
<th>MCST</th>
<th>MIS</th>
<th>Cond.</th>
<th>SpMV</th>
<th>Pagerank</th>
<th>BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon0601</td>
<td>0.61s</td>
<td>1.12s</td>
<td>0.83s</td>
<td>0.37s</td>
<td>3.31s</td>
<td>0.07s</td>
<td>0.09s</td>
<td>0.25s</td>
<td>1.38s</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>2.98s</td>
<td>0.69s</td>
<td>0.29s</td>
<td>2.35s</td>
<td>3.72s</td>
<td>0.19s</td>
<td>0.19s</td>
<td>0.74s</td>
<td>6.32s</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>7.22s</td>
<td>11.12s</td>
<td>9.60s</td>
<td>7.66s</td>
<td>15.54s</td>
<td>0.78s</td>
<td>0.74s</td>
<td>2.90s</td>
<td>1m 21s</td>
</tr>
<tr>
<td>dimacs-usa</td>
<td>6m 12s</td>
<td>9m 54s</td>
<td>38m 32s</td>
<td>4.68s</td>
<td>9.60s</td>
<td>0.26s</td>
<td>0.65s</td>
<td>2.58s</td>
<td>12.01s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ssd</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Friendster</td>
<td>38m 38s</td>
<td>1h 8m 12s</td>
<td>1h 57m 52s</td>
<td>1h 16m 29s</td>
<td>2m 3s</td>
<td>3m 41s</td>
<td>15m 31s</td>
<td>52m 24s</td>
<td></td>
</tr>
<tr>
<td>sk-2005</td>
<td>44m 3s</td>
<td>1h 56m 58s</td>
<td>2h 13m 5s</td>
<td>19m 13s</td>
<td>2m 14s</td>
<td>1m 59s</td>
<td>8m 9s</td>
<td>56m 29s</td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>19m 19s</td>
<td>35m 23s</td>
<td>32m 25s</td>
<td>10m 17s</td>
<td>47m 43s</td>
<td>1m 40s</td>
<td>1m 29s</td>
<td>6m 12s</td>
<td>42m 52s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>disk</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Friendster</td>
<td>1h 17m 18s</td>
<td>2h 29m 39s</td>
<td>3h 53m 44s</td>
<td>43m 19s</td>
<td>4m 25s</td>
<td>7m 42s</td>
<td>32m 16s</td>
<td>1h 57m 36s</td>
<td></td>
</tr>
<tr>
<td>sk-2005</td>
<td>1h 30m 3s</td>
<td>4h 40m 49s</td>
<td>4h 41m 26s</td>
<td>39m 12s</td>
<td>4m 45s</td>
<td>4m 12s</td>
<td>17m 22s</td>
<td>2h 24m 28s</td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>39m 47s</td>
<td>1h 39m 9s</td>
<td>1h 10m 12s</td>
<td>29m 8s</td>
<td>3m 38s</td>
<td>3m 13s</td>
<td>13m 21s</td>
<td>2h 8m 13s</td>
<td></td>
</tr>
<tr>
<td>yahoo-web</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
# Algorithmic Performance

<table>
<thead>
<tr>
<th></th>
<th>WCC</th>
<th>SCC</th>
<th>SSSP</th>
<th>MCST</th>
<th>MIS</th>
<th>Cond.</th>
<th>SpMV</th>
<th>Pagerank</th>
<th>BP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>memory</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>amazon0601</td>
<td>0.61s</td>
<td>1.12s</td>
<td>0.83s</td>
<td>0.37s</td>
<td>3.31s</td>
<td>0.07s</td>
<td>0.09s</td>
<td>0.25s</td>
<td>1.38s</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>2.98s</td>
<td>0.69s</td>
<td>0.29s</td>
<td>2.35s</td>
<td>3.72s</td>
<td>0.19s</td>
<td>0.19s</td>
<td>0.74s</td>
<td>6.32s</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>7.22s</td>
<td>11.12s</td>
<td>9.60s</td>
<td>7.66s</td>
<td>15.54s</td>
<td>0.78s</td>
<td>0.74s</td>
<td>2.90s</td>
<td>1m 21s</td>
</tr>
<tr>
<td>dimacs-usa</td>
<td><strong>6m 12s</strong></td>
<td><strong>9m 54s</strong></td>
<td><strong>38m 32s</strong></td>
<td>4.68s</td>
<td>9.60s</td>
<td>0.26s</td>
<td>0.65s</td>
<td>2.58s</td>
<td>12.01s</td>
</tr>
<tr>
<td>ssd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friendster</td>
<td>38m 38s</td>
<td>1h 8m 12s</td>
<td>1h 57m 52s</td>
<td>1h 16m 29s</td>
<td>2m 3s</td>
<td>3m 41s</td>
<td>15m 31s</td>
<td>52m 24s</td>
<td></td>
</tr>
<tr>
<td>sk-2005</td>
<td>44m 3s</td>
<td>1h 56m 58s</td>
<td>2h 13m 5s</td>
<td>19m 30s</td>
<td>3h 21m 18s</td>
<td>2m 14s</td>
<td>1m 59s</td>
<td>8m 9s</td>
<td>56m 29s</td>
</tr>
<tr>
<td>Twitter</td>
<td>19m 19s</td>
<td>35m 23s</td>
<td>32m 25s</td>
<td>10m 17s</td>
<td>47m 43s</td>
<td>1m 40s</td>
<td>1m 29s</td>
<td>6m 12s</td>
<td>42m 52s</td>
</tr>
<tr>
<td><strong>disk</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friendster</td>
<td>1h 17m 18s</td>
<td>2h 29m 39s</td>
<td>3h 53m 44s</td>
<td>43m 19s</td>
<td>2h 39m 16s</td>
<td>4m 25s</td>
<td>7m 42s</td>
<td>32m 16s</td>
<td>1h 57m 36s</td>
</tr>
<tr>
<td>sk-2005</td>
<td>1h 30m 3s</td>
<td>4h 40m 49s</td>
<td>4h 41m 26s</td>
<td>39m 12s</td>
<td>7h 1m 21s</td>
<td>4m 45s</td>
<td>4m 12s</td>
<td>17m 22s</td>
<td>2h 24m 28s</td>
</tr>
<tr>
<td>Twitter</td>
<td>39m 47s</td>
<td>1h 39m 9s</td>
<td>1h 10m 12s</td>
<td>29m 8s</td>
<td>1h 42m 14s</td>
<td>3m 38s</td>
<td>3m 13s</td>
<td>13m 21s</td>
<td>2h 8m 13s</td>
</tr>
<tr>
<td>yahoo-web</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>16m 32s</td>
<td>14m 40s</td>
</tr>
</tbody>
</table>
Algorithmic Performance

<table>
<thead>
<tr>
<th>Graph</th>
<th># steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-memory</td>
<td></td>
</tr>
<tr>
<td>amazon0601</td>
<td>19</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>20</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>15</td>
</tr>
<tr>
<td>dimacs-usa</td>
<td>8122</td>
</tr>
<tr>
<td>Out-of-core</td>
<td></td>
</tr>
<tr>
<td>sk-2005</td>
<td>28</td>
</tr>
<tr>
<td>yahoo-web</td>
<td>over 155</td>
</tr>
</tbody>
</table>

- HyperANF measures the *neighbourhood function* of the graph.
- High numbers indicate those graphs have a **large diameter**.
- They take many scatter-gather iterations to complete, which is why X-Stream performs poorly on them.
Scalability
X-Stream vs Graphchi

<table>
<thead>
<tr>
<th>Twitter pagerank</th>
<th>Pre-Sort (s)</th>
<th>Runtime (s)</th>
<th>Re-sort (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Stream (1)</td>
<td>none</td>
<td>397.57 ± 1.83</td>
<td>–</td>
</tr>
<tr>
<td>Graphchi (32)</td>
<td>752.32 ± 9.07</td>
<td>1175.12 ± 25.62</td>
<td>969.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Netflix ALS</th>
<th>Pre-Sort (s)</th>
<th>Runtime (s)</th>
<th>Re-sort (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Stream (1)</td>
<td>none</td>
<td>76.74 ± 0.16</td>
<td>–</td>
</tr>
<tr>
<td>Graphchi (14)</td>
<td>123.73 ± 4.06</td>
<td>138.68 ± 26.13</td>
<td>45.02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RMAT27 WCC</th>
<th>Pre-Sort (s)</th>
<th>Runtime (s)</th>
<th>Re-sort (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Stream (1)</td>
<td>none</td>
<td>867.59 ± 2.35</td>
<td>–</td>
</tr>
<tr>
<td>Graphchi (24)</td>
<td>2149.38 ± 41.35</td>
<td>2823.99 ± 704.99</td>
<td>1727.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Twitter belief prop.</th>
<th>Pre-Sort (s)</th>
<th>Runtime (s)</th>
<th>Re-sort (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Stream (1)</td>
<td>none</td>
<td>2665.64 ± 6.90</td>
<td>–</td>
</tr>
<tr>
<td>Graphchi (17)</td>
<td>742.42 ± 13.50</td>
<td>4589.52 ± 322.28</td>
<td>1717.50</td>
</tr>
</tbody>
</table>

- X-Stream outperforms Graphchi:
  - No pre-processing cost.
  - X-Stream makes a better use of the SSD, constantly maintaining high bandwidth.
X-Stream vs Graphchi

- X-Stream outperforms Graphchi:
  - No pre-processing cost.
  - X-Stream makes a better use of the SSD, constantly maintaining high bandwidth.
Review

• X-Stream achieved impressive results against existing solutions.
• Points attention to the trade-off between number and cost of memory accesses.
• On the downside, X-Stream’s performance is heavily dependant on the characteristics of the underlying graph.
Impact

• **Chaos** is the next generation of X-Stream.

• A couple of studies have adopted X-Stream’s edge-centric approach:
  • “An FPGA framework for edge-centric graph processing” (S. Zhou et al.)
  • “WolfGraph: The edge-centric graph processing on GPU” (H. Zhu et al.)

• However, the application of edge-centric frameworks seems to be restricted to academia.
Thank you for the attention!