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

Amitabha Roy, Ivo Mihailovic, Willy Zwaenepoel
Context
Approach
Model
Implementation
Results & Conclusion
Pregel & Powergraph: scatter & gather

→ A scatter-gather methodology:

1. scatter(vertex v):
   send updates over outgoing edges of v

2. gather(vertex v):
   apply updates from inbound edges of v

→ how to scale-up?
**Trade-off:** Sequential vs Random access

<table>
<thead>
<tr>
<th>Medium</th>
<th>Read (MB/s)</th>
<th>Write (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM (1 core)</td>
<td>Random 567</td>
<td>Random 1057</td>
</tr>
<tr>
<td>RAM (16 cores)</td>
<td>Sequential 2605</td>
<td>Sequential 2248</td>
</tr>
<tr>
<td>SSD</td>
<td>Random 14198</td>
<td>Sequential 10044</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>Sequential 25658</td>
<td>Random 48.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sequential 576.5</td>
</tr>
<tr>
<td></td>
<td>22.5</td>
<td>48.6</td>
</tr>
<tr>
<td></td>
<td>667.69</td>
<td>576.5</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>328</td>
<td>316.3</td>
</tr>
</tbody>
</table>

**Figure 11: Sequential Access vs. Random Access**
GraphChi: a sequential approach

→ avoids random access using *shards*

Problems:

1. need graph to be pre-sorted by source vertex
2. vertex-centric
3. requires re-sort of edges by destination vertex for gather step
Context
Approach
Model
Implementation
Results & Conclusion
X-Stream’s Approach

1. retain *scatter-gather* programming model
2. use an edge-centric implementation
3. stream unordered edge lists

**Gains:**

1. use sequential (*not* random) access
2. do not need pre-processing step
scatter-gather: an edge-centric implementation

scatter(edge e):
    send update over e

gather(update u):
    apply update u to u.destination
Quick Terminology

Fast Storage:
→ caches (in-memory)
→ main-memory (out-of-core)

Slow Storage:
→ main-memory (in-memory)
→ SSD/Disk (out-of-core)
Context
Approach
Model
Implementation
Results & Conclusion
The basic model:

*input*: an unordered set of directed edges

*API*: implementations of scatter/gather for given edges
Problem: vertices may *not* fit in fast storage

1. Edge Centric Scatter
   Edges (sequential read)

   Vertices (random read/write)

   Updates (sequential write)

2. Edge Centric Gather
   Updates (sequential read)

   Vertices (random read/write)
Problem: vertices may *not* fit in fast storage

→ Streaming partitions:

- **vertex set**, $V$: a subset of the vertices of the graph
- **edge list**: source is $\in V$
- **update list**: dest $\in V$

→ How do we use them?

1. scatter/gather iterate over streaming partitions
2. updates need to be *shuffled*
Context
Approach
Model
Implementation
Results & Conclusion
Stream buffer

Index Array (K entries)

Chunck Array

Chunck

Chunck

Chunck
Out-of-core

→ *Folds shuffle into scatter*
  - run scatter, appending updates to an in-memory buffer
  - when buffer full: run an *in-memory shuffle*

→ *2 Stream Buffers*

→ *Number of partitions*

\[ \frac{N}{K} + 5SK \leq M \]

→ *Disk I/O*

In-memory

→ *Parallel multi-stage shuffler & scatter/gather*
  - stream independently for each streaming partition
  - work stealing
  - group partitions together into a tree for the shuffler

→ *3 stream buffers*

→ *Number of partitions*

= CPU_cache_size / footprint
Chaos: the extension of X-Stream

→ Scale out to multiple machines in 1 cluster

2 gains:

1. access secondary storage in parallel improves performance
2. increases size of graph that can be handled
Chaos: the extension of X-Stream

→ Steps:

1. simple initial partitioning
2. spread graph data uniformly over all 2nd storage devices
3. work stealing

Assumptions:

1. network machine-to-machine bandwidth > bandwidth of storage device
2. network switch bandwidth > aggregate bandwidth of all storage devices of cluster
Context
Approach
Model
Implementation
Results & Conclusion
### Experiments:

→ Tested on real-world graphs.

<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>

| SSD     | Friendster | 38m 38s | 1h 8m 12s | 1h 57m 52s | 19m 13s | 1h 16m 29s | 2m 3s | 3m 41s | 15m 31s | 52m 24s |
|         | sk-2005    | 44m 3s  | 1h 56m 58s | 2h 13m 5s  | 19m 30s | 3h 21m 18s | 2m 14s | 1m 59s | 8m 9s   | 56m 29s |
|         | Twitter    | 19m 19s | 35m 23s   | 32m 25s   | 10m 17s | 47m 43s   | 1m 40s | 1m 29s | 6m 12s  | 42m 52s |

| Disk    | Friendster | 1h 17m 18s | 2h 29m 39s | 3h 53m 44s | 43m 19s | 2h 39m 16s | 4m 25s | 7m 42s | 32m 16s | 1h 57m 36s |
|         | sk-2005    | 1h 30m 3s  | 4h 40m 49s | 4h 41m 26s | 39m 12s | 7h 1m 21s | 4m 45s | 4m 12s | 17m 22s | 2h 24m 28s |
|         | Twitter    | 39m 47s   | 1h 39m 9s  | 1h 10m 12s | 29m 8s  | 1h 42m 14s | 3m 38s | 3m 13s | 13m 21s | 2h 8m 13s |
|         | yahoo-web  | —         | —         | —         | —      | —         | 16m 32s | 14m 40s | 1h 21m 14s | 8h 2m 58s |
Scalability
Comparison

---

**RMAT scale 25 graph**

- WCC
- Pagerank
- BFS
- SpMV

**Processing time (s)**

<table>
<thead>
<tr>
<th>Threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**BFS on scale-free graph (32M vertices/256M edges)**

- Local Queue
- Hybrid
- X-Stream

**Runtime (s)**

<table>
<thead>
<tr>
<th>Threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Comparison: Ligra

<table>
<thead>
<tr>
<th>Threads</th>
<th>Ligra (s)</th>
<th>X-Stream (s)</th>
<th>Ligra-pre (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11.10</td>
<td>168.50</td>
<td>1250.00</td>
</tr>
<tr>
<td>2</td>
<td>5.59</td>
<td>86.97</td>
<td>647.00</td>
</tr>
<tr>
<td>4</td>
<td>2.83</td>
<td>45.12</td>
<td>352.00</td>
</tr>
<tr>
<td>8</td>
<td>1.48</td>
<td>26.68</td>
<td>209.40</td>
</tr>
<tr>
<td>16</td>
<td>0.85</td>
<td>18.48</td>
<td>157.20</td>
</tr>
<tr>
<td>Pagerank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>990.20</td>
<td>455.06</td>
<td>1264.00</td>
</tr>
<tr>
<td>2</td>
<td>510.60</td>
<td>241.56</td>
<td>654.00</td>
</tr>
<tr>
<td>4</td>
<td>269.60</td>
<td>129.72</td>
<td>355.00</td>
</tr>
<tr>
<td>8</td>
<td>145.40</td>
<td>83.42</td>
<td>211.40</td>
</tr>
<tr>
<td>16</td>
<td>79.24</td>
<td>50.06</td>
<td>160.20</td>
</tr>
</tbody>
</table>
Comparison: Graphchi

<table>
<thead>
<tr>
<th></th>
<th>Pre-Sort (s)</th>
<th>Runtime (s)</th>
<th>Re-sort (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Twitter pagerank</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
<tr>
<td><strong>Netflix ALS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
<tr>
<td><strong>RMAT27 WCC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
<tr>
<td><strong>Twitter belief prop.</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
Conclusion & Takeaway

**Strengths:**

→ Sequential access
→ Scale up & scale out

**Weaknesses**

→ Limited number of problems it can handle
→ Limited types of graphs it can handle
→ How would you use in a real-world scenario