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

Authors: Amitabha Roy, Ivo Mihailovic, Willy Zwaenepoel
Why Previous Solutions Do Not Scale

Random Access

Sequential Access

Vertex-Centric

Edge-Centric
**Vertex-Centrism**

- Requires edge and vertex data in fast memory
- Allows for pre-processing(sorting) edge data for faster algorithms

**Problems**

- Most Graphs have significantly more edges than vertices
- Harder to partition graph data
- Random access over both vertices and edges
- Pre-processing dominates the run-time

**Scatter**($v : \text{vertex}$):
  
  $\text{Send(Outgoing}[V])$

**Gather**($v : \text{vertex}$):
  
  $\text{Apply(Incoming}[V])$
**Edge-Centrism**

Scatter(e: edge): Send_Update(e)

Gather(u :update): Apply(u, u.dest)

1. Edge Centric Scatter
   Edges (sequential read)

2. Edge Centric Gather
   Updates (sequential read)

- Requires almost zero pre-processing
- Only needs fast random access to vertex data

**Benefits**

- Better mapping to hardware and the structure of real graphs
- Allows for streaming the edge data from slow memory sequentially, speed-up:
  - Disk: 500x
  - SSD: 300x
  - RAM: 4.6x/1.8x for 1/16 cores
- Better initial run-time performance
- No bottlenecks in maintaining invariants
Streaming Partitions

Basic mapping to hardware:
- Vertex set should fit into fast storage
- Assume uniform distribution of updates
- Divide the graph uniformly
  - Considering auxiliary data structures
  - Algorithms are order-independent

Vertex Set

Edge List

Scatter Phase

Shuffle Phase

Gather Phase
Two types of relative storage: Slow vs Fast

Partitioning the Memory Hierarchy:

- Fast: Cache, Ram
- Slow: Disk

Hierarchical Memory Processing

- X-Stream implements streaming engines for handling transfer from slow to fast storage
  - Making heavy use of large static streaming buffers to carry data

In-Memory Streaming Engine

Out-of-Core Streaming Engine
Out-of-Core Streaming Engine

Moves data from disk to memory
• In order to transfer data from disk to memory it uses a simple stream buffer
• Partition size and buffer are both statically allocated

Modified computation model
• All incoming/outgoing data from/to disk passes through in/out buffers (2 of each for prefetching)
• Shuffle stages are performed within scatter phases whenever UOut becomes full
  • Maximizes buffer usage
**In-Memory Streaming Engine**

**Implementation Details**

**Parallel Multistage Shuffler:**
- Arranges partitions into a tree structure
- Uses a power of two for both the number of partitions and fanout
- Inputs get shuffled by being passed down the tree and split up at every step

**Layered Approach:**
- Sits above the disk streaming layer
- Disk layer operates as normal, however the in-memory processing of a partition is further fed into the in-memory system
Performance Evaluation Results

Wasted Computation:

<table>
<thead>
<tr>
<th></th>
<th># iters</th>
<th>ratio wasted</th>
<th>wasted %</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td>19</td>
<td>2.58</td>
<td>63</td>
</tr>
<tr>
<td>amazon0601</td>
<td>21</td>
<td>2.20</td>
<td>50</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>13</td>
<td>2.13</td>
<td>57</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>6263</td>
<td>1.94</td>
<td>98</td>
</tr>
</tbody>
</table>

| ssd      |         | 1.06         | 63       |
| friendster | 24      | 1.04         | 63       |
| sk-2005  | 25      | 1.04         | 67       |
| twitter  | 16      | 1.04         | 55       |
| yahoo-web | —       | —            | —        |

This is a direct result of a large-diameter structure.

Wasted computation is an expected trade-off from large-scale streaming.

Sorting dominates the run-time of most systems we will see.

System scales very well on most tasks, linearly until a new storage medium is needed.
Ligra And Graphchi Comparison

- Pre-processing in Ligra takes longer than the entire X-Stream execution.

<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>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>
</tbody>
</table>

Figure 20: Ligra [48] on Twitter (99% CI under 5%)

Graphchi serves a similar use case to X-Stream while applying a vertex-centric model. The average speed-up without pre-processing is 2.3 and 3.7 with pre-processing.

- Disk bandwidth usage is also more predictable in X-Stream.

Overall, Ligra should still massively overperform on speed in most real use-cases.

Figure 21: Instructions per Cycle and Total Number of Memory References for BFS

The efficiency of sequential memory access also makes X-Stream dominate in IPC.

Figure 22: Comparison with Graphchi on SSD with 99% Confidence Intervals. Numbers in brackets indicate X-Stream streaming partitions/Graphchi shards (Note: ressorting is included in Graphchi runtime).
Opinion/Motivation

Can Sorting Keep-Up?
- Any vertex-centric computation requires some way of associating edges to source/destination vertices, sorting is the most popular
- Sometimes it is necessary to look at a reversed edge-list for classes of algorithms
  - Requires either re-sorting repeatedly or maintaining two views of the edge list.
- This narrative has been extensively challenged by Frank McSherry using radix sort to process twitter data 10x faster than the X-Stream authors estimation

- Vertex-Centric: Edge Data/RAM Bandwidth
- Edge-Centric: Scatter X E_Data/Seq Band

Real-World Graphs:
- All of the scale-free graphs perform very well with X-Stream, many real-world graphs follow a power-law distribution.
- Work stealing seems sufficient to handle high-degree vertices.
- Real world graphs grow very slowly in diameter $O(\log(V)/\log(\log(V)))$ and can even undergo densification
Context

Creation:
• According to Amitabha Roy in “X-Stream: A Case Study in Building a Graph Processing System”
• The algorithms used within the system were first devised by observing the relation between graph processing and sparse matrix-vector multiplication
  • Followed by applying advances in SpMV to graph processing
• The implementation, systems and evaluation were subsequently developed for publishing in “Symposium on Operating Systems Principles”
  • The final paper changes the focus to the systems aspect of X-Stream

Development History:
• The GitHub has not had any commits in years
• Authors from EPFL also developed “Chaos” as the multi-machine successor of X-Stream, utilizing many of the same ideas surrounding streaming partitions with a heavy focus on work stealing and ignoring locality
  • The new system is capable of handling graphs with 1 trillion edges ~ 16 TB of data
  • Later scaled to 8 trillion on only 32 machines
• “Chaos” development, at least publicly, also seems to have ceased soon after creation
Why Has It Not Had a Larger Impact?

- Unusual edge-centric computational model
- Algorithmic origins:
  - Tumultuous implementation
  - Potentially difficult to extend or maintain
- No way of easily changing graph structure, relies on static data structures
- No long-term support
- Lacks comprehensive documentation, high-level means of integration, or a killer-app
- Highly focused towards throughput and cost over speed, niche use-case
  - As shown by Frank McSherry, for some specific tasks, better algorithms implemented with less-restrictive programming models and efficient pre-processing may be superior.
- In my opinion, it was never intended for production
- As an academic work it has a fair number of citations and inspired systems
- Similar critiques apply to “Chaos”
Questions!