Scale-up Graph Processing: A Storage-centric View

Eiko Yoneki University of Cambridge

Amitabha Roy EPFL
Graph Storage

- Storing and accessing graphs is a challenge
  - Edge traversal produces an access pattern that is
    - Random
    - Unpredictable

- For scale up or limited scale out (small clusters)
  - Storage bottlenecks (RAM, SSD, Magnetic disk) in critical path
RASP and X-Stream

- Storage-centric: two different novel ways to access graph structured data
  - Batch processing of large graphs on single machine
  - Establish useful limits for single machine processing
  - Directly address storage bottlenecks

RASP: Accelerates random access using a novel prefetcher

X-Stream: Sequentially streaming a large set of (potentially unrelated) edges

- RASP and X-stream take (diametrically opposite) storage centric view of graph processing problems
RASP: Run Ahead SSD Prefetcher

- Prefetching allows cheap hardware to compete with supercomputing for suitable graph traversal
- Prefetcher ensures that edge data to progress computation is always available in memory
- Allows graph traversal to keep queue depth high → SSD to achieve good performance
- Vertices (O(V)) size structure in memory
- Edges in SSD in CSR format
- Efficient on traversal: WCC, BFS, SSSP, A*...
**Edge Queue Management**

- Prefetcher invokes any registered callbacks, accessing the current state of the main program’s iterator
- Asynchronous page load requests to OS via fadvise
- Repeat to ensure future data to active LRU list

![Diagram of Edge Queue Management](image-url)
RASP Speedup

- Speedups from up to 13x comparing over single and multithreaded versions
- RASP Memory usage WCC

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
<th>RAM</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter [10]</td>
<td>52M</td>
<td>1.6B</td>
<td>1.18GB</td>
<td>8.4GB</td>
</tr>
<tr>
<td>Scale-free-small [12]</td>
<td>32M</td>
<td>1.28B</td>
<td>1.10GB</td>
<td>12GB</td>
</tr>
<tr>
<td>Scale-free-large [12]</td>
<td>1B</td>
<td>40B</td>
<td>24GB</td>
<td>315GB</td>
</tr>
</tbody>
</table>

- RASP Runtime (mins) for WCC

<table>
<thead>
<tr>
<th>Graph</th>
<th>Base</th>
<th>RASP</th>
<th>MT(8)</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter</td>
<td>36.08</td>
<td>6.36</td>
<td>7.17</td>
<td>2.31</td>
</tr>
<tr>
<td>Erdos-Reyni</td>
<td>80.96</td>
<td>6.03</td>
<td>11.30</td>
<td>4.11</td>
</tr>
<tr>
<td>Scale-free-small</td>
<td>88.56</td>
<td>10.77</td>
<td>15.74</td>
<td>3.95</td>
</tr>
<tr>
<td>Scale-free-large</td>
<td>&gt;2.5 days</td>
<td>402.56</td>
<td>&gt;2 days</td>
<td>cannot fit</td>
</tr>
</tbody>
</table>
Vertex/Edge Centric Access

- Vertex centric access is random
- Edge centric access is more sequential
- Can subdivide into streaming partitions
X-Stream: Streaming Partitions

- Sequential access to any medium
- Eliminate random access to edges
- Ensure randomly accessed vertices held in cache
Scale-up with X-stream

- Scaling up through RAM, SSD and Magnetic Disk

X-Stream Breadth-first Traversal

- 2B vertices/32B edges/23 hours
- 128M vertices/2B edges/26 mins
- 8M vertices/256M edges/23 sec

Lower is better

Graph size (Vertices/Edges)
(run with 16 cores machines)
Pros and Cons

- **RASP** clearly provides impressive speedup
- Improving inefficiency of random access to SSD by prefetching

**Limitation**
- RASP requires pre-processing to CSR format
- RASP is specific to SSD
- Focus is traversal based graph computation (not for DFS)

- **X-Stream** transforms them to sequential access
- Single building block of streaming partitions
  - Works well with RAM, SSD, and Magnetic Disk

**Limitation**
- X-stream needs to trade off fewer random accesses to edge list for sequential bandwidth of streaming a large number of potentially unrelated edges
RASP+X-Stream Hybrid Approach

- Allow streaming partitions to sort their associated edges and access them randomly
  - Starting point is X-stream style streaming
  - Low utilization of edges due to few active vertices triggers index building
    - Switch to RASP style prefetching after index is available
  - Streaming partition has the necessary vertex subset in memory: a requirement for RASP
  - RASP mitigates limitations of X-Stream
    - Wasted edges due to inactive vertices
    - Particular problem for high diameter graphs
IVEC Programming Model

- Abstract interface for graph algorithms, we intend to support
- Iterative Vertex-Centric programming model
  - Scatter: Vertex state → updates along edges
  - Gather: Updates on incoming edges → vertex state
  - IVEC can be mapped to Pregel, Powergraph, Graphchi ...
  - GreenMarl (optimised iterative operation)

- Can express variety of graph operations
  - BFS/WCC, SSSP, PageRank, MIS...
  - But not algorithms with O(E) state, such as triangle counting
- Hides complexity of algorithms and storage from each other
Conclusion

- Storing and accessing graphs is a challenge since it is determinant for performance in graph processing
- RASP and X-stream: address diametrically opposite storage centric view of graph processing problem
  - **RASP**: Accelerates random access with prefetcher
  - **X-Stream**: Sequentially streaming edges
- Towards hybrid approach of RASP + X-Stream
- Scale out for bandwidth and capacity
  - Target 1T edges
- Explore 'limited' scale out
  - Network does not become the bottleneck
  - Multiply storage capacity, bandwidth and compute
  - Tightly coupled solutions: micro servers, low power boards