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

Amitabha Roy, Ivo Mihailovic, Willy Zwaenepoel (EPFL - SOSP’13)

Presented by: Amir H. Payberah
amir@sics.se
Feb. 7, 2014
Graphs

Presented by: Amir H. Payberah (SICS)
Large graphs are a subset of the big data problem.

Billions of vertices and edges, hundreds of gigabytes.

Normally tackled on large clusters.

- Pregel, Giraph, GraphLab, PowerGraph ...
- Complexity, power consumption ...
Could we compute Big Graphs on a single machine?
Challenges

- **Disk-based processing**
  - *Problem*: graph traversal = random access
  - Random access is inefficient for storage

---

Challenges

▶ Disk-based processing
  - Problem: graph traversal = random access
  - Random access is inefficient for storage

<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</td>
<td>567</td>
<td>2605</td>
</tr>
<tr>
<td>SSD</td>
<td>22.64</td>
<td>355</td>
</tr>
<tr>
<td>Disk</td>
<td>0.61</td>
<td>174</td>
</tr>
</tbody>
</table>

Note: 64 byte cachelines, 4K blocks (disk random), 16M chunks (disk sequential)

Solution

X-Stream makes graph accesses sequential.
Edge-centric scatter-gather model

Streaming partitions
Edge-Centric Scatter-Gather Model
State stored in vertices.

Vertex operations:
- Scatter updates along outgoing edges
- Gather updates from incoming edges
Iterates over vertices

```plaintext
for each vertex v
    if v has update
        for each edge e from v
            scatter update along e
```
for each vertex v
  if v has update
    for each edge e from v
      scatter update along e
for each vertex v
    if v has update
        for each edge e from v
            scatter update along e
for each vertex $v$
  if $v$ has update
    for each edge $e$ from $v$
      scatter update along $e$
for each vertex v
  if v has update
    for each edge e from v
      scatter update along e
for each vertex v
    if v has update
        for each edge e from v
            scatter update along e
Vertex-Centric vs. Edge-Centric Access

Vertex-centric

- Vertices
- Edges
- Sequential
- Random

Edge-centric

- Vertices
- Edges
- Sequential
- Random
Edge-Centric Scatter

- Iterates over edges

```java
for each edge e
    if e.src has update
        scatter update along e
```
for each edge e

if e.src has update
    scatter update along e
for each edge $e$
    if $e.src$ has update
        scatter update along $e$
for each edge e
    if e.src has update
        scatter update along e
for each edge e
    if e.src has update
        scatter update along e
for each edge e
    if e.src has update
        scatter update along e
Vertex-Centric vs. Edge-Centric Tradeoff

- **Vertex-centric** scatter-gather:  \( \frac{\text{EdgeData}}{\text{RandomAccessBandwidth}} \)

- **Edge-centric** scatter-gather:  \( \frac{\text{Scatters} \times \text{EdgeData}}{\text{SequentialAccessBandwidth}} \)

- Sequential Access Bandwidth ➪ Random Access Bandwidth.

- Few scatter gather iterations for real world graphs.
Streaming Partitions
Problem: still have random access to vertex set.
Problem

- Problem: still have random access to vertex set.

Solution

Partition the graph into streaming partitions.
Partitioning the Graph (1/2)

vertices

```
<table>
<thead>
<tr>
<th>v1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>
```

edges

```
<table>
<thead>
<tr>
<th>src</th>
<th>dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>v2</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>src</th>
<th>dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Random access for free.
Streaming Partition

- A subset of the vertices that fits in RAM.

- All edges whose source vertex is in that subset.

- No requirement on quality of the partition, e.g., sorting edges.

- Consists of three sets: vertex set, edge list, and update list.
Streaming Partition Scatter-Gather

- The scatter phase iterates over all streaming partitions, rather than over all edges.

- The gather phase iterates over all streaming partitions, rather than over all updates.

- The vertex sets and edge lists remain fixed during the entire computation.

- The update list of a partition varies over time.
// Scatter phase
for each streaming_partition p {
    read in vertex set of p
    for each edge e in edge list of p
        edge_scatter(e): append update to Uout
}

// Shuffle phase
for each update u in Uout {
    let p = partition containing target of u
    append u to Uin(p)
}
destroy Uout

// Gatter phase
for each streaming_partition p {
    read in vertex set of p
    for each update u in Uin(p)
        edge_gather(u): apply update u to u.destination
    destroy Uin(p)
}
- X-Stream

- Scatter-Gather model

- Edge-centric: sequential access to the graph edges

- Streaming partition: vertex set, edge list, and update list
Questions?

Acknowledgement

Some slides were derived from the slides of Amitabha Roy (EPFL)