NXgraph: An Efficient Graph Processing System on a Single Machine

Yuze Chi\textsuperscript{1}, Guohao Dai\textsuperscript{1}, Yu Wang\textsuperscript{1}, Guangyu Sun\textsuperscript{2}, Guoliang Li\textsuperscript{1} and Huazhong Yang\textsuperscript{1}

Tsinghua National Laboratory for Information Science and Technology\textsuperscript{1}, Tsinghua University
Center for Energy Efficient Computing and Applications\textsuperscript{2}, Peking University
Motivation

Big Data → Structure → Big Graph

- Social media: Facebook, Twitter
- Science: DNA, protein structure
- Advertising: Amazon, Netflix, eBay
- Web: Google, Wikipedia, Bing

2016.06.05
Motivation

- Computation capacity of a single CPU
  - Intel i7-5820K: 12 hyper-threads, 3.3GHz
  - Assume 50 clock cycles/edge: **792 MEPS** (Million Edges Per Second)

<table>
<thead>
<tr>
<th>System</th>
<th>Throughput/MEPS (PageRank on Twitter)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark\textsuperscript{HotCloud2010}</td>
<td>15 in total/0.15 each</td>
<td>100 CPUs in 50 nodes</td>
</tr>
<tr>
<td>PowerGraph\textsuperscript{OSDI2012}</td>
<td>408 in total/6.4 each</td>
<td>64 CPUs in 64 nodes</td>
</tr>
<tr>
<td>GraphChi\textsuperscript{OSDI2012}</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>TurboGraph\textsuperscript{SIGKDD2013}</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>X-stream\textsuperscript{SOSP2013}</td>
<td>20</td>
<td>no pre-processing</td>
</tr>
<tr>
<td>VENUS\textsuperscript{ICDE2015}</td>
<td>15.4</td>
<td>on HDD</td>
</tr>
<tr>
<td>GridGraph\textsuperscript{ATC2015}</td>
<td>61</td>
<td></td>
</tr>
</tbody>
</table>

- Large gap!
- Our objective: **higher MEPS/CPU**

2016.06.05
Vertex-centric Model

- Graph $G = (V,E)$
  - vertex $v = (id, attributes)$
  - edge $e = (src \_vid, dst \_vid, additional \_attribute)$
- "Think like a vertex"

- Example: Breadth-First Search (BFS)
  
  ```plaintext
  for each dst in my.out_edges
    if dst.depth > my.depth+1
      then
        dst.depth = my.depth+1
  ```
• Vertices → Intervals
• Edges → Shards
  – Edges in each shard → Sub-Shards

• Objective
  – Limit memory access to a small region
  – Improve locality
# Four Optimizing Rules

<table>
<thead>
<tr>
<th></th>
<th>GraphChi</th>
<th>TurboGraph</th>
<th>VENUS</th>
<th>GridGraph</th>
<th>NXgraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Reduce the amount of disk data transfer</td>
<td></td>
<td>:green_emoji:</td>
<td>:green_emoji:</td>
<td>:green_emoji:</td>
<td>:green_emoji:</td>
</tr>
</tbody>
</table>

No previous system considers them all
Our Effort

- Follows all four optimizing rules

- According to optimizing rule 1 & 2, we design
  - Destination-Sorted Sub-Shard (DSSS) structure

- According to optimizing rule 3 & 4, we design
  - Adaptive updating strategies
    - Single-Phase Update (SPU)
    - Double-Phase Update (DPU)
    - Mixed-Phase Update (MPU)
- Sort edges in each sub-shard
- Sequential reads from source interval (Rule 1)
  - Potentially improves cache hit rate
- Parallel writes to destination vertices (Rule 2)
  - No write conflict among threads
- Defines behavior **inside** each sub-shard
Single-Phase Update

Always resident in memory (Rule 3)

source intervals

destination intervals

I₁
0, 1

S₁
SS₁₁
1→2
0, 1→3

I₂
2, 3

S₂
SS₁₂
SS₂₁
2, 3→1

I₃
4, 5

S₃
SS₁₃
SS₂₂
SS₂₃

I₄
6

S₄
SS₁₄
SS₂₄
SS₃₄
SS₄₄

Iᵢ
vertices

In-memory ping-pong interval

SSᵢᵢ
sorted edges

What if memory is insufficient?

Streamed from disk (Rule 4)
Double-Phase Update (To-Hub Phase)

I$_i$ vertices

On-disk interval

SS$_{ij}$ edges

Sub-shard

Hubs now hold all incremental information

Execution order

Load from disk

I$_1$

0, 1

I$_2$

2, 3

I$_3$

4, 5

I$_4$

6

S$_1$

SS$_{1,1}$

S$_2$

SS$_{1,2}$

S$_3$

SS$_{1,3}$

S$_4$

SS$_{1,4}$

SS$_{2,1}$

SS$_{2,2}$

SS$_{2,3}$

SS$_{2,4}$

SS$_{3,1}$

SS$_{3,2}$

SS$_{3,3}$

SS$_{3,4}$

SS$_{4,1}$

SS$_{4,2}$

SS$_{4,3}$

SS$_{4,4}$

2016.06.05
Double-Phase Update (From-Hub Phase)

Save to disk

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>$I_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1</td>
<td>2, 3</td>
<td>4, 5</td>
<td>6</td>
</tr>
</tbody>
</table>

$S_1$

$SS_{1,1}$

$SS_{2,1}$

$SS_{3,1}$

$SS_{4,1}$

$S_2$

$SS_{1,2}$

$SS_{2,2}$

$SS_{3,2}$

$SS_{4,2}$

$S_3$

$SS_{1,3}$

$SS_{2,3}$

$SS_{3,3}$

$SS_{4,3}$

$S_4$

$SS_{1,4}$

$SS_{2,4}$

$SS_{3,4}$

$SS_{4,4}$

On-disk interval

$SS_{ij}$

edges

Sub-shard with a hub

Execution order

Memory space is not fully utilized

2016.06.05
### Mixed-Phase Update

**Always resident in memory**

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td></td>
<td>SS_{1.1}</td>
<td>SS_{1.2}</td>
<td>SS_{1.3}</td>
<td>SS_{1.4}</td>
</tr>
<tr>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Load from disk**

<table>
<thead>
<tr>
<th>$I_3$</th>
<th>$I_4$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,5</td>
<td></td>
<td>SS_{3.1}</td>
<td>SS_{3.2}</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>SS_{3.3}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td></td>
<td>SS_{1.1}</td>
<td>SS_{1.2}</td>
<td>SS_{1.3}</td>
<td>SS_{1.4}</td>
</tr>
<tr>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SS_{2.1}</td>
<td>SS_{2.2}</td>
<td>SS_{2.3}</td>
<td>SS_{2.4}</td>
</tr>
<tr>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SS_{3.1}</td>
<td>SS_{3.2}</td>
<td>SS_{3.3}</td>
<td>SS_{3.4}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Save to disk**

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td></td>
<td>SS_{1.1}</td>
<td>SS_{1.2}</td>
<td>SS_{1.3}</td>
<td>SS_{1.4}</td>
</tr>
<tr>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SS_{2.1}</td>
<td>SS_{2.2}</td>
<td>SS_{2.3}</td>
<td>SS_{2.4}</td>
</tr>
<tr>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SS_{3.1}</td>
<td>SS_{3.2}</td>
<td>SS_{3.3}</td>
<td>SS_{3.4}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hub attached to a sub-shard**

**Fully utilize available memory**

---

**2016.06.05**
System Architecture

NXgraph

preprocessing

indexer

sharder

updater

indexer

- generate vertex mapping
- generate binary edge list
- calculate vertex degree

sharder

- put edges into each shard
- sort edges in each shard

updater

- calculate output in a user-defined format with user-defined input

2016.06.05
Results

- Evaluation platform
  - Hex-core hyper-threading Intel i7-5820K CPU @ 3.3GHz
  - 8x8G DDR4 RAM, 2x128G RAID0 SSD, 1T HDD
  - Ubuntu 14.04 LTS 64bit/Windows 10 Edu 64bit
Results: Design Decisions

Performance with different sub-shard model

<table>
<thead>
<tr>
<th>Model</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Live-journal</td>
</tr>
<tr>
<td>src-sorted, coarse-grained</td>
<td>1.44s</td>
</tr>
<tr>
<td>dst-sorted, fine-grained</td>
<td>1.00s</td>
</tr>
</tbody>
</table>

Destination-sorted is the right choice!

Performance with different numbers of interval on Twitter

~10s intervals are near-optimal choices

2016.06.05
Results: Design Decisions

SPU vs DPU on performance

Always prefer SPU over DPU

PageRank

BFS

PageRank

BFS

SCC

2016.06.05
Results: Different Environments

PageRank on Live-journal

PageRank on Twitter

PageRank on Yahoo-web

Up to 13.64x speedup over TurboGraph

PageRank on Live-journal

PageRank on Twitter

PageRank on Yahoo-web

2016.06.05
Results: Scalability and More Tasks

Scalability

Best scalability

More tasks on Twitter

Average speedup of 74.6x

More tasks on Live-journal

More tasks on Yahoo-web

2016.06.05
## Results: More Systems (Limited Resources)

### System performance with limited resources

<table>
<thead>
<tr>
<th>System</th>
<th>Time (s)</th>
<th>Speedup</th>
<th>Evaluation environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXgraph</td>
<td>7.13</td>
<td>1.00</td>
<td>Intel i7 3.3GHz, 8t, 8G, SSD</td>
</tr>
<tr>
<td>GridGraph</td>
<td>26.91</td>
<td>3.77</td>
<td>AWS EC2 8t, 8G/30.5G, SSD</td>
</tr>
<tr>
<td>X-stream</td>
<td>88.95</td>
<td>12.48</td>
<td>AWS EC2, 8t, 8G/30.5G, HDD</td>
</tr>
<tr>
<td>NXgraph</td>
<td>12.55</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>VENUS</td>
<td>95.48</td>
<td>7.60</td>
<td></td>
</tr>
<tr>
<td>GridGraph</td>
<td>24.11</td>
<td>1.92</td>
<td></td>
</tr>
<tr>
<td>X-stream</td>
<td>81.70</td>
<td>6.51</td>
<td></td>
</tr>
</tbody>
</table>

Average speedup of **6.6x** over various state-of-the-art single-machine systems with limited resources.

Task: 1 iteration of PageRank on Twitter graph

2016.06.05
Results: More Systems (Best Case)

System performance in the best case

<table>
<thead>
<tr>
<th>System</th>
<th>Time (s)</th>
<th>Speedup</th>
<th>Evaluation environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXgraph</td>
<td>2.05</td>
<td>1.00</td>
<td>Intel i7 3.3GHz, 8t, 16G, SSD</td>
</tr>
<tr>
<td>X-stream</td>
<td>23.25</td>
<td>11.57</td>
<td>AMD 6400, 24G, SSD</td>
</tr>
<tr>
<td>GridGraph</td>
<td>24.11</td>
<td>11.99</td>
<td>AMD 6400, 24G, SSD</td>
</tr>
<tr>
<td>MMAP</td>
<td>13.10</td>
<td>6.52</td>
<td>AMD 6400, 24G, SSD</td>
</tr>
<tr>
<td>PowerGraph</td>
<td>3.60</td>
<td>1.79</td>
<td>Dell T6600, 24G, SSD</td>
</tr>
</tbody>
</table>

Task: 1 iteration of PageRank on Twitter graph

717MEPS actual throughput vs 792MEPS hypothetical limit
Future Work

- NXgraph is still under development and subject to changes in data structures and APIs

v0.2 (current)
- Faster sharding (done)
- More flexible partitioning (done)
- Faster indexing (active)
- More complicated algorithms (active)

v0.3 (scheduled)
- Dynamic graph support (scheduled)
- Open source (scheduled: mid August)

Timeline:
- Asynchronous updating (scheduled)
- More intelligent partitioning (pending)
- Search for practical applications (pending)

Contact us if you would like to see a pre-release!

2016.06.05
Reference

Reference

- Yahoo! altavisata web page hyperlink connectivity graph, circa 2002,”
Thank you!

Q&A