RealGraph: A Graph Engine Leveraging the Power-Law Distribution of Real-World Graphs

Yong-Yeon Jo, Myung-Hwan Jang, Sang-Wook Kim, Sun-Ju Park

Presenter: Zhi Wei Gan
Current Graph Engines

• Follows a vertex-centric programming model

• Graph engines adopt iterative processing:
  
  1. Identify nodes to be processed
     • Scan indicator (bit vector with length equal to number of nodes)
  
  2. Process identified nodes and edges
     • Threads process nodes and edges in parallel (Node-based workload allocation)

Indicator: 010101

1 2 + Edges

3 4 + Edges

5 6 + Edges

Thread 1

Thread 2

Thread 3
Power Law Recap

- Many vertices have few edges
- Few vertices have many edges
- Most graph processing frameworks process one vertex a thread
- Some iterations of the algorithm will process nodes with a very large difference in degree, so threads processing vertices with fewer edges will be waiting for other vertices with more edges to complete
Figure 1: Degree distribution of the Yahoo dataset.

Figure 2: Cumulative number of the nodes processed up to each iteration when BFS is performed.

Figure 3: Degree difference between the nodes processed at each iteration when BFS is performed.
What problem does RealGraph solve?

• Existing graph engines use indicators to identify vertices to process
  • This looks like a single linear indicator
  • RealGraph introduces the Hierarchical Indicator to solve this problem

• The uneven workload allocation over threads leads to waiting threads
  • Using block-based workload allocation solves this.
  • Vertices aren’t given to threads, instead blocks are
RealGraph Architecture

Web interface is provided for ease of use

Performs the graph algorithm. Thread pool: several worker threads and one main thread that maintains workers
Attribute data: stores results for nodes and edges. One element per node in several vectors.
Indicator: same as before

Maintains index that maps object id to block location

Manages blocks stored in main memory

Manages storage space and divides graph data into blocks. Objects that exceed block size are stored in multiple blocks
Efficient Layout

• RealGraph assigns consecutive node IDs to nodes likely to be accessed by the same/successive iterations by graph algorithms

• Order is defined by a BFS that is run once before user-provided algorithms are run

  • 4 blocks instead of 8!
  • Empty circles are just unlabeled

Figure 5: Data layout transformation.
Hierarchical Indicator

• Simple idea that improves on the current linear indicator by using a tree structure to avoid unnecessary scanning
• Identifies nodes to be processed in a top-down manner
• Next-iteration indicator populated in a bottom-up manner

Range length: 3
Height: 2

The height/range length can be tuned as parameters to each dataset

Figure 6: 3-level hierarchical indicator.
Parameter tuning on Yahoo Dataset

Range length: \(2^{10}\) is best
Block-based Workload Allocation

- Instead of processing vertices on threads, we process blocks

- What if objects are larger than 1 block?
  - Object stored across multiple blocks and is handled by multiple threads
  - Thread synchronization is required in this case **(Slow! Overhead! Bad!)**
  - Only 0.0003% of objects stored in several blocks on average with block size 1024KiB
Experiments

• Regular single-machine graphs.
• Intel i7-7700K (4 cores 8 threads), 1TiB SSD and 64GiB Memory
• Evaluation of paper and techniques used
• Comparisons with single machine graph engines
• Comparisons with distributed system graph engines
## Evaluation of Improvements

### Key:
- **H**: Hierarchical Indicator
- **D**: Data Layout Transformation
- **B**: Block-based Workload
- **Ω**: All three

### Figure 10: Performance of proposed techniques.

(a) BFS

<table>
<thead>
<tr>
<th>Combinations</th>
<th>Execution time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>1,149</td>
</tr>
<tr>
<td>H</td>
<td>757</td>
</tr>
<tr>
<td>B</td>
<td>591</td>
</tr>
<tr>
<td>D</td>
<td>279</td>
</tr>
<tr>
<td>HB</td>
<td>115</td>
</tr>
<tr>
<td>HD</td>
<td>35</td>
</tr>
<tr>
<td>BD</td>
<td>253</td>
</tr>
<tr>
<td>Ω</td>
<td>19</td>
</tr>
</tbody>
</table>

(b) PageRank

<table>
<thead>
<tr>
<th>Combinations</th>
<th>Execution time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>3,236</td>
</tr>
<tr>
<td>B</td>
<td>2,434</td>
</tr>
<tr>
<td>D</td>
<td>747</td>
</tr>
<tr>
<td>Ω</td>
<td>385</td>
</tr>
</tbody>
</table>
Comparison with vertex-centric & external-memory graph engines

![Graph performance comparison charts for BFS, WCC, and PageRank](chart.png)

Figure 11: Performance comparison with graph engines with the same model.
Comparison single-machine graph engines with other models

Figure 12: Performance comparison with graph engines with the different models.

X-Stream: edge-centric, external memory. FlashGraph: vertex-centric, semi-external memory.
RealGraph performs better than everything else other than PowerGraph. Lots of overhead with synchronization of distributed systems.

It would be interesting to see how RealGraph performs on graphs that do not follow the Power Law.