Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets

Nadathur Satish, Narayanan Sundaram, Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M. Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey

Presented by Guoyao Feng
Agenda

• Introduction
• Graph Algorithms
• Graph Analytics Frameworks
• Experimental Setup
• Experiment Results
• Optimizations and Recommendations
• Conclusion
• Discussion
Introduction: Background

• Growing interest in creating, storing and *processing large graph data*
Introduction: Motivation

- Graph algorithm implementation
  - Irregular computation
  - Resource under-utilization
  - Large performance gap: Naive implementation vs. hand-optimized code
- No standard “building block”
  - Sparse matrix, vertex-centric programming, etc.
- Performance varies depending on both frameworks and algorithms
  - A headache to choose frameworks

Create a roadmap to improve graph frameworks’ performance
Bridge the performance gap against native code
Graph Algorithms

- **PageRank**
  - Iteratively computes rank (web page popularity) for each vertex (web page) in a directed graph (reference web)
  
  \[ PR^{t+1}(i) = r + (1 - r) \times \sum_{j \mid (j, i) \in E} \frac{PR^t(j)}{\text{degree}(j)} \]

- **Breadth Frist Search (BFS)**
  - Traverses an undirected, unweighted graph from one vertex and compute the minimal distance
  - In each iteration:
    
    \[ \text{Distance}(i) = \min_{j \mid (j, i) \in E} \text{Distance}(j) + 1 \]

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Graph Algorithms

• **Triangle Counting**
  • Each pair of vertices in an edge compare their neighbourhood lists and count the number of shared neighbours

\[
N_{\text{triangles}} = \sum_{i,j,k; i < j < k} E_{ij} \land E_{jk} \land E_{ik}
\]

• **Collaborative Filtering**
  • Estimates the rating of an item by a given user

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Graph Analytics Frameworks: GraphLab

- Graph algorithms expressed as programs running on a vertex
- Each vertex reads incoming messages, updates states and sends message asynchronously

**PageRank**

Algorithm 1: Vertex program for one iteration of page rank

```
begin
    \[ PR \leftarrow r \]
    for \( msg \in \text{incoming messages} \) do
        \[ PR \leftarrow PR + (1 - r) \times msg \]
    Send \( \frac{PR}{\text{degree}} \) to all outgoing edges
```

**BFS**

Algorithm 2: Vertex program for one iteration of BFS.

```
begin
    for \( msg \in \text{incoming messages} \) do
        \[ Distance \leftarrow \min(Distance, msg + 1) \]
    Send Distance to all outgoing edges
```

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Graph Analytics Frameworks: CombBLAS

- Provides linear algebra primitives for graph analytics
- Operates on sparse matrix and vectors
- Edge-based partitioning (2-D partitioning)

**PageRank**

\[ p_{t+1} = r \mathbf{1} + (1 - r) \mathbf{A}^T \tilde{\mathbf{p}}_t \]

- Page rank values at iteration \( t+1 \)
- Adjacency matrix

**BFS**

\[ \mathbf{v} = \mathbf{A}^T \mathbf{s} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 1 \end{pmatrix} \]

- Vector of starting vertices
- Next vertices to explore

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Graph Analytics Frameworks: SociaLite

- Declarative language running recursive queries
- Horizontally partitioned for parallelism
- **PageRank**

\[
\text{Rank}[n](t+1, \Sigma \text{Sum}(v)) : v = r \\
: \text{InEdge}[n](s), \text{Rank}[s](t, v_0), \text{OutDeg}[s](d), v = \frac{(1 - r)v_0}{d}
\]

- **Triangle Counting**

\[
\text{Triangle}(0, \Sigma \text{Inc}(1)) : \neg \text{Edge}(x, y), \text{Edge}(y, z), \text{Edge}(x, z)
\]

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Graph Analytics Frameworks: Giraph

- Bulk synchronous graph processing system on Hadoop
- Vertex partitioning (1-D partitioning)

**Collaborative Filtering**
- Gradient Descent
- In one iteration, every vertex
  1. Aggregates information from neighbours
  2. Sends updated vector to neighbours

\[
p_u^* = p_u + \gamma t \sum_{v|(u,v)\in E} [R_{uv}q_v - (p_u^T q_v)q_v - \lambda_p p_u]
\]

\[
q_v^* = q_v + \gamma t \sum_{u|(u,v)\in E} [R_{uv}p_u - (p_u^T q_v)p_u - \lambda_q q_v]
\]

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Graph Analytics Frameworks: Galois

- Framework designed for irregular computation
- Work-item based parallelization
- Automatous scheduling and scalable data structures
- Runs on a single node
- **Triangle Counting**

```
Algorithm 4: Galois program for Triangle counting.

begin
    Graph G
    numTriangles = 0
    foreach (Node n: G) in parallel do
        S₁ = { m in G.neighbors(n) | m > n }
        for (m in S₁) do
            S₂ = { p in G.neighbors(m) | p > m }
            numTriangles ← numTriangles + |S₁ ∩ S₂|

end
```

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
## Experimental Setup

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Vertices</th>
<th># Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook [1]</td>
<td>2,937,612</td>
<td>41,919,708</td>
</tr>
<tr>
<td>Wikipedia [2]</td>
<td>3,566,908</td>
<td>84,751,827</td>
</tr>
<tr>
<td>Twitter [4]</td>
<td>17,770 movies</td>
<td></td>
</tr>
<tr>
<td>Yahoo Music [5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synthetic Graph500</td>
<td>536,870,912</td>
<td>8,589,926,431</td>
</tr>
<tr>
<td>Synthetic Collaborative Filtering</td>
<td>63,367,472 users</td>
<td>16,742,847,256 ratings</td>
</tr>
<tr>
<td></td>
<td>1,342,176 items</td>
<td></td>
</tr>
</tbody>
</table>
Experiment Results: Native Code

- Native hand-optimized implementation efficiency

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Single Node</th>
<th></th>
<th>4 Nodes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H/W limitation</td>
<td>Efficiency</td>
<td>H/W limitation</td>
<td>Efficiency</td>
</tr>
<tr>
<td>PageRank</td>
<td>Memory BW</td>
<td>78 GBps (92%)</td>
<td>Network BW</td>
<td>2.3 GBps (42%)</td>
</tr>
<tr>
<td>BFS</td>
<td>Memory BW</td>
<td>64 GBps (74%)</td>
<td>Memory BW</td>
<td>54 GBps (63%)</td>
</tr>
<tr>
<td>Coll. Filtering</td>
<td>Memory BW</td>
<td>47 GBps (54%)</td>
<td>Memory BW</td>
<td>35 GBps (41%)</td>
</tr>
<tr>
<td>Triangle Count.</td>
<td>Memory BW</td>
<td>45 GBps (52%)</td>
<td>Network BW</td>
<td>2.2 GBps (40%)</td>
</tr>
</tbody>
</table>

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Experiment Results: Single Node

- Performance on a single node with real world and synthetic graphs

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Experiment Results: Multiple Nodes

- Performance on multiple nodes using large synthetic graphs

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Experiment Results: Multiple Nodes

- Performance on multiple nodes using large synthetic graphs

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Experiment Results: Summary

• Slowdown factors of framework performance against native code on a *single node*

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CombBLAS</th>
<th>GraphLab</th>
<th>SociaLite</th>
<th>Giraph</th>
<th>Galois</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank</td>
<td>1.9</td>
<td>3.6</td>
<td>2.0</td>
<td>39.0</td>
<td>1.2</td>
</tr>
<tr>
<td>BFS</td>
<td>2.5</td>
<td>9.3</td>
<td>7.3</td>
<td>567.8</td>
<td>1.1</td>
</tr>
<tr>
<td>Coll. Filtering</td>
<td>3.5</td>
<td>5.1</td>
<td>5.8</td>
<td>54.4</td>
<td>1.1</td>
</tr>
<tr>
<td>Triangle Count.</td>
<td>33.9</td>
<td>3.2</td>
<td>4.7</td>
<td>484.3</td>
<td>2.5</td>
</tr>
</tbody>
</table>

• Slowdown factors of framework performance against native code on *multiple nodes*

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CombBLAS</th>
<th>GraphLab</th>
<th>SociaLite</th>
<th>Giraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank</td>
<td>2.5</td>
<td>12.1</td>
<td>7.9</td>
<td>74.4</td>
</tr>
<tr>
<td>BFS</td>
<td>7.1</td>
<td>29.5</td>
<td>18.9</td>
<td>494.3</td>
</tr>
<tr>
<td>Coll. Filtering</td>
<td>3.5</td>
<td>7.1</td>
<td>7.0</td>
<td>87.9</td>
</tr>
<tr>
<td>Triangle Count.</td>
<td>13.1</td>
<td>3.6</td>
<td>1.5</td>
<td>54.4</td>
</tr>
</tbody>
</table>

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Experiment Results: Framework Analysis

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Optimizations

• Key optimizations in native implementation
  • Data structures
  • Data compression
  • Overlap of Computation and Communication
  • Message passing mechanisms
  • Partitioning schemes

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"
Recommendations

• **GraphLab**
  • Mainly limited by network bandwidth ⇒ MPI
  • Data compression, prefetching, computation and communication overlap

• **CombBLAS**
  • Use bit-vector for compression in BFS
  • Techniques for inter-operation optimization

• **Galois**
  • Implemented most optimizations

• **Giraph**
  • Boost network bandwidth
  • Data compression
  • Reduce memory buffer size for higher memory efficiency

• **SociaLite**
  • Most algorithms limited by network bandwidth
  • Data compression
Conclusion

• Compares graph frameworks in terms of programming model and implementation of multiple algorithms
• Exposes performance gap (2-30X) between graph frameworks and hand-optimized native code
• Analyzes CPU usage, memory footprint, and network traffic to explain performance gap
• Shows performance gains of optimization techniques in native code and recommendations for graph frameworks

“our goal is not to come up with a new graph processing benchmark or propose a new graph framework, but to analyze existing approaches better to find out where they fall short”
Discussion

• The optimization techniques are known when the native code is implemented. Why not apply them directly to the frameworks if possible?

• The paper analyzes framework in terms of CPU usage, memory footprint, and network traffic. How can we reason about the performance difference based on the programming models?
  • For example, vertex programming vs. parallel graph library

• What are the pros and cons of ...
  • Using only one graph framework
  • Selecting the framework to use based on the algorithm
  • Simply developing the native implementations
References