GraphBench: Towards a Universal Graph Benchmark

Khaled Ammar
M. Tamer Özsu
Bioinformatics

Gene Co-expression

Protein Structure

Software Engineering

Social Network

Program Flow
Big Graphs

- Twitter breaks 400 million tweets per day
- 2.4 billion internet users as of June-2012
- 200+ million new Facebook users annually since 2009
- 40+ billion web pages indexed by Google
- 300 million new websites in 2011
- RedHat 7.1 has 30+ million line of code (60% more than Redhat 6.2)
Graph Processing Systems

PREGEL

Project Pegasus

Hadoop MapReduce

HaLoop

GraphLab

UNIVERSITY OF WATERLOO
Problems!

• Systems report their performance with comparison to Hadoop MapReduce only!

• Systems usually use PageRank algorithm only!
A solution

A Benchmark with the following conditions:

• Represent all query types
• Generate graphs with real-life properties
• Easy to implement by existing systems
Workload

- Online Queries
- Update Queries
- Iterative Queries
Workload - Online Queries

• Find matching nodes/edges
• Find k-hop neighbours
• Reachability and shortest path query
• Pattern matching query
Workload - Update Queries

- Insert / Update / Delete an edge or a node

- The Caveat:
  - Update queries might seem straightforward but in reality they are challenging because they may require an update to a pre-computed answer or an index.
Workload - Iterative Queries

These are Data Mining Algorithms

- Examples include PageRank, Clustering… etc
- The main theme:
  - "iterate on graph data until convergence".
Data Generator

Based on RTG generator which extended Millar’s work in power-law distribution generation.
Data Generation: Real-life Graph properties

- Very Large
- Exponential growth in size
- Sparse
- Small diameters
- Community structure
- One large connected component
- Power Law degree distribution
PowerLaw generation

Random process to generate PowerLaw distribution:

• A Monkey press random keys on a keyboard
• One character is selected to separate words
• The distribution of the generated words is PowerLaw.
1. Initiate a symmetric probability matrix for all words’ characters and the escape character ϕ.

2. Characters in the main row makes the source vertex.

3. Characters in the main column makes the destination vertex.
## Data Generator: RTG

### RTG parameters:

- Number of characters
- Probability of $\phi$ (escape char)
- Structure factor (0-1)
- Graph Weight (# edges)
- Bipartite flag
- Number of time stamps (Dynamic graphs)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>...</th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\phi$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Generator : RTG

RTG parameters:

- Number of characters
- Probability of $\phi$ (escape char)
- Structure factor (0-1)
- Graph Weight (# edges)
- Bi-Partite flag
- Number of time stamps (Dynamic graphs)

It is challenging to fit these parameters!
Data Generator

Caveat: Some nodes and edges are going to be identical.

\[ N \propto W^{-\log_p k} \]

\( N = \) Number of unique nodes
\( k = \) Number of characters
\( q = \) Probability of \( \phi \) \((q = 1 - k \times p)\)
\( p = \) Probability of a character
\( W = \) Graph Weight (Total number of edges)
\( E = \) Number of unique edges

\[ E \approx W^{-\log_p k} \times (1 + c' \log W), \quad \text{for} \quad c' = \frac{q^{-\log_p k}}{-\log p} > 0 \]
Data Generator

\[ N \propto W^{-\log_p k} \]

\( N = \) Number of unique nodes
\( k = \) Number of characters
\( q = \) Probability of \( \phi \) (\( q = 1 - k \times p \))
\( p = \) Probability of a character
\( W = \) Graph Weight (Total number of edges)
\( E = \) Number of unique edges

\[ E \approx W^{-\log_p k} \times (1 + c' \log W), \quad \text{for} \quad c' = \frac{q^{-\log_p k}}{-\log p} > 0 \]

Graph Density:

\[ D = \frac{E}{N \times N - 1} \]
Data Generator

$N = \text{Number of unique nodes}$

$k = \text{Number of characters}$

$q = \text{Probability of } \phi \ (q = 1 - k \times p)$

$p = \text{Probability of a character}$

$W = \text{Graph Weight (Total number of edges)}$

$E = \text{Number of unique edges}$

$E \approx W^{-\log_p k} \times (1 + c' \log W), \text{ for } c' = \frac{q^{-\log_p k}}{-\log p} > 0$

Graph Density $D = F_n (E, N) = F_n (p, k, W, q) = F_n (q, k, W)$
Data Generator

**Target:** Generate graphs matching specific graph properties

**How:** Replace two parameters \((k, q)\) by other graph properties
Data Generator

Target: Generate graphs with properties
How: Replace two parameters ($k, q$) by graph properties
Observation: for a reasonable “$k$”, number of characters in the language does not impact graph properties.
Data Generator

Observation: for a reasonable “k”, number of characters in the language does not impact graph properties.

Graph Density

\[ D = F_n (q, k, W) = F_n (q, W) \]
Data Generator

Given a reasonable $k (=10)$, Find $q$

$$D = Fn (q, W) \Rightarrow q = Fn (D, W)$$

*This is a Complex and approximate equation*
Data Generator

Find $q$ using supervised learning techniques.

- We generated a large number of graphs with different settings and used a Gaussian model to predict $q$.

- Based on our experiments, the average prediction error was 10%
Conclusion

• Graphs are emerging in many applications
• Graph processing systems need a benchmark
• Benchmark should be complete, efficient, and usable.
• G Benchmark has graph queries that represents all workload types.
• G Benchmark has a usable real-graph generator
What is next?

• Build an extensive quantitative comparison between existing Graph systems

• Integrate our graph generators with business driven data generators like ones in TPC-H or BigBench.
Thanks
Existing BenchMarks

HPC Scalable Graph Analysis Benchmark
Graph Traversal Benchmark
Graph500
BigBench
RDF Benchmarks