HPC Graph Analytics on the OneGraph Model

LDBC TUC 2023
June 24, 2023

Ümit V. Çatalyürek
Amazon Scholar, Amazon Web Services
Professor, School of Computational Science and Engineering
Georgia Institute of Technology

Brad Bebee, Dave Bechberger, Willem Broekema, Hal Cooper, Olaf Hartig, Ankesh Khandelwal, Ora Lassila, Kelvin Lawrence, Carlos Manuel Lopez Enriquez, Michael Schmidt, Ronak Sharda, Lefteris Sidirourgos, Gabriel Tanase, Bryan Thompson, and Gregory Williams

Neptune Team, Amazon Web Services
Graphs are Ubiquitous and Fun!

They are growing. Up to billions of vertices and edges

Fast, efficient analysis is important and pervasive

Many graph processing frameworks, and databases, have been proposed/developed

**Image credits:**
- Jenn Caulfield, Social network vector illustration, 2018
- Gerhard et al., Frontiers in Neuroinformatics 5(3), 2011
- Albert-László Barabási/BarabasiLab 2019
- Caleb Jonson, How to Visualize Your Twitter Network, 2014
Landscape of current “Graph World”

Enterprise Graph Frameworks

Graph Databases

HPC Graph Analytics

24 June 2023 @ LDBC TUC
Scalability

Our goal is to have a performance that is in small-constant factor from HPC / State-of-the-art Graph Analytics, yet provide easy to maintain and productive development environment.

**LPG: Labeled Property Graphs**

**Vertices**
Nodes: Label/ID + *Properties* (set of key-value pairs)

**Edges**
Relationships: Label/ID + Type + Properties

**Example**
- **NID1** (name: "Umit")
- **NID2** (name: "Kaan")
- **EID1** (relation: "son")

**RDF: Resources Description Framework**

**Vertices**
Resources: URIs
Attribute Values: Literals

**Edges**
Relationships: URIs

**Example**
- **S** (subject)
- **O** (object)
- **P** (predicate)

**RDF Triple**: Subject-Predicate-Object

There is no internal structure for nodes and edges
Graph interoperability

- Amazon Neptune
  - managed, cloud-based graph database service
  - supports RDF (SPARQL) and LPG (Gremlin & openCypher)
- User has to choose either RDF or LPG
  - this choice also determines which query languages are available
  - the choice is not always easy, and is hard to reverse later
- RDF vs. LPG
  - RDF offers a formal model, LPG not so much
  - RDF is “sometimes seen as academic”, and developers tend to prefer LPG
  - different strengths and weaknesses
Graph interoperability

- What if we did not have to choose between RDF and LPG?
- What if we could use Gremlin over RDF, or SPARQL over LPG?
- Interoperability: single graph (meta)model, free use of any query language
  - we are not interested in “qualified” interoperability where one meta-model is implemented using the other
- RDF-star is a step towards having LPG features in RDF

- 1G model ("one graph to rule them all")
Storage Challenges: Interoperability

- Interoperability: serve both RDF and LPG
- 1G Graph Storage
  - Three kinds of relations
    - Dictionaries: URIs/Literals → IDs
    - Graph Structure: Topologies – relations between (S)ubject and (O)bject, in other words between “vertices”
    - Graph Data: Values – properties of vertices and edges
  - In 1G, Edges/Properties (of vertices and edges) can become “vertices”
  - Relations are partitioned (sharded)
    - 1D: Dictionaries, Vertex Properties etc.
    - 2D: Topology and properties of edges (collocated for performance)
Storage Challenges: Dynamic Partitioned Data

- Graph is not static (well, obviously!)
  - Many HPC Graph Analytics kernels assumes graph is not changing.
  - Even dynamic ones conveniently ignores deletion.

- How to (dynamically) distribute data?

- System generated IDs are uniform random
  - Notice that graph comes as vertices as URIs
  - Load-balanced partitioning (declustering) is favored against locality for initial load
  - Graph-aware re-ordering/re-labeling can be done after graph is loaded

- Sharding options: Node partition (1D) vs Edge partition (fine-grain 2D) vs Blocked partition (coarse-grain 2D)
  - Blocked partition is used as a sweet spot between performance and architecture agnostic algorithm development [1] (more on next slide)

---

[1] PGAbB: A Block-Based Graph Processing Framework for Heterogeneous Platforms
Abdurrahman Yasar, Sivasankaran Rajamanickam, Jonathan W. Berry, Umit V. Catalyurek
https://arxiv.org/abs/2209.04541
Why 2D?: PGAbB Results on Selected Graphs [1]

- Power9 (2 x 16 x 4) CPUs & Volta100 GPU.
- 320 GB Host Memory. 32 GB Device Memory.
- CPU-GPU bandwidth: ~60GB/s
- PGAbB: Kokkos at the backend with OpenMP (Host) and Cuda (Device)
- All normalzed wrt GAPBS

### Table 1.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
<th>Triangles</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter7</td>
<td>41.6 M</td>
<td>1.2 B</td>
<td>34.8 B</td>
<td>0.001</td>
</tr>
<tr>
<td>Com-Orkut</td>
<td>3 M</td>
<td>117 M</td>
<td>627 M</td>
<td>0.041</td>
</tr>
<tr>
<td>Sk-2005</td>
<td>50.6 M</td>
<td>1.8 B</td>
<td>84.9 B</td>
<td>0.002</td>
</tr>
<tr>
<td>Kmer V1r</td>
<td>214 M</td>
<td>232 M</td>
<td>49</td>
<td>0.000</td>
</tr>
<tr>
<td>Europe-OSM</td>
<td>50.9 M</td>
<td>54.1 M</td>
<td>61 K</td>
<td>0.003</td>
</tr>
<tr>
<td>Myciel.19</td>
<td>393 K</td>
<td>451 M</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kron-Scale21</td>
<td>2.1 M</td>
<td>91 M</td>
<td>8.8 B</td>
<td>0.044</td>
</tr>
</tbody>
</table>

How about computational model

- Internally we provide
  - From “think like a vertex” to “think like a sub-graph (block/tile)”
  - Visitor model

- Externally:
  - Currently openCypher + with Graph API

- Ümit says
  - Low-hanging fruit: GraphBLAS
Storage Challenges: Scalability and Transactions

- **Scalability: Scaling Up (vertical/single-node) and Scaling Out (horizontal/multi-node)**
  - Read scaling is “easy”
  - Write scaling with transaction support is challenging:
    - Distributed in-memory graph storage with logging is still challenging to implement.

- **What does it mean to provide Graph Analytics under transactional system?**
  - Transaction aware reads
    - Index-driven vs Scan-based kernels and dynamic tradeoffs based on cardinality estimates
    - Dynamic creation of “Views” for multi-iteration algorithms
Effect of Shading to Performance

- **Early results** on a single, old EC2 instance
- Each shard executed sequentially (no fine-grain parallelism)
- Results show expected behavior:
  - Performance of BFS correlated with avg degree
  - 2D partitioning/sharding is not ideal for BFS, especially for very sparse data, but works well for almost all others, such as PageRank

<table>
<thead>
<tr>
<th>name</th>
<th>#vertices</th>
<th>#edges (undirected)</th>
<th>avg degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-LiveJournal</td>
<td>3,997,962</td>
<td>34,681,189</td>
<td>17.35</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>3,072,441</td>
<td>117,185,083</td>
<td>76.28</td>
</tr>
<tr>
<td>europe_osm</td>
<td>50,912,018</td>
<td>54,054,660</td>
<td>2.12</td>
</tr>
</tbody>
</table>

"HPC Graph Analytics on the OneGraph Model"
Computational Infrastructure Challenges

- Can we implement once, and run everywhere: from multi-core to multi-host with potentially accelerators?
  - Yes!
    - Multi-Level Intermediate Representation (MLIR) for Graphs
    - “Coarse-grained” Labeled-Dataflow Execution

- Can we support both OLAP and OLTP graph data management?
  - Yes!
    - Native Storage
    - Advanced Scan Kernels
    - State-of-the-art Transactional Model (MVCC, …)
Conclusions & Future Directions

- HTAP (i.e., Hybrid OLTP and OLAP) solutions are needed!
  - Enterprise Graph Systems gives the *illusion* of read scaling, while failing in absolute performance, and write/update scaling (they just leave that to IO system)
  - HPC Graph Analytics codes/libraries, are one-off, focused on narrow set of kernels and fail to provide end-to-end solutions
  - Existing “Real” Graph Databases, provides either OLTP or OLAP, but fails to deliver both

- Interoperability is a big challenge!
  - SPARQL, Gremlin and OpenCypher queries for both OLTP and OLAP workloads

- Graph as a Service

- It is exciting times for Graphs!