Graphs in Big Data: Challenges and Opportunities

Yinglong Xia
05/16/2016

Mission-Critical Big Data Analytics (MCBDA’2016)
Graph is the way we remember, we associate, and we understand.
Background
Graph Analytics and Systems
Challenges
Breakthrough & Opportunities
Mini Hands-on
Classic Graph Theory

- In 1736, Seven Bridges of Königsberg is historically proposed in mathematics, laid the foundations of graph theory.

- In 1878, Graph theory is discussed by Sylvester in *Nature*.

- The first textbook on graph theory was written by Dénes König in 1936, followed by another one by Frank Harary in 1969.
Brief History

N.T. Bliss, Confronting the Challenges of Graphs and Networks, Lincoln Laboratory Journal, 2013

Neuronal network @ Human Brain Project 89 billion V & 100 trillion E

61.6 million V 1.47 billion E

40 million V 300 million E

N. T. Bliss, Confronting the Challenges of Graphs and Networks, Lincoln Laboratory Journal, 2013
Diversity in Graph Technology

**Dynamic graph** helps analyze the spatial and temporal influence over the entities in the network.

**RDF graph** enables knowledge inference over linked data.

**Streaming graph** monitors sentiment propagation over time and how the graph structure can impact.

**Property graph** is widely used as a data storage model to manage the properties of entities as well as the interconnections.

Graph technology leads to rich analytic abilities.

**Graphical models** leverages statistics to inference latent factors in a complex system.
Graphs in Big Data

- CDR graph: Call detailed record can form a graph by linking the numbers called each other.
- Social network is a scale-free graph with small-world effect.
- Some recommender system such as collaborative filter can be constructed on a bipartite graph.
- Graphical Models can be used to find latent variables from noisy data.
Graph Analytics
Complex Network Analysis

Real world **complex networks** include WWW, Social Network, Biological network, Citation Network, Power Grid, Food Web, Metabolic network, etc.

**Import properties/metrics:**
- Small-world effect
- Betweenness
- Eccentricity/Centrality
- Transitivity
- Resilience
- Community structure
- Clustering coefficient
- Matching index

**Complex network models:**
- Poisson random graph
  - degree~Poisson
  - Small world effect
- Watts and Strogatz graph
  - Transitivity
  - Small world effect
- Barabasi and Albert graph
  - Small world
  - Power law
Information Propagation

Graph showing the propagation of information over time with different line graphs representing various scenarios.

Graph legend:
- K-EdgeAdd
- CompDeg
- CompEigs
- CompDelete
- Rand
- CompPage
- Original

Time Ticks: 0 to 5000
Log (Infected Ratio) scale: -6 to 2.5

Initial Node
Knowledge Graph

- RDF
  - Represent relationships among entities using links with properties
  - W3C/DAWG Standards

RDF is a key part of semantic network, making the WWW into a info exchange media

<table>
<thead>
<tr>
<th>subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yinglong</td>
<td>work_in</td>
<td>Futurewei</td>
</tr>
<tr>
<td>Yinglong</td>
<td>born</td>
<td>1980</td>
</tr>
<tr>
<td>Futurewei</td>
<td>has_HQ</td>
<td>Shenzhen</td>
</tr>
</tbody>
</table>

RDF Graph = A collection of triples, linking the description of resources

SPARQL is the standard language to query graph data represented as RDF triples
Graphical Model for Probabilistic Inference

- **Graphical Model**
  - Represent joint distribution of r.v. compactly using the conditional independence among factors
  - Components:
    - node → random variable
    - edge → prob dependence
  - Examples
    - Bayesian Network
    - Latent Markov Field
    - Factor Graph
    - Boltmann Machine

  **Use case**: Computer vision, image processing

---

**Bayesian Network 101**

**Random variable**

**Dependence**

**CPT**

**Joint Distribution**

\[ P(G, S, R) = P(G | S, R)P(S | R)P(R) \]

**Probabilistic Inference**: Inferring the status of the unobservable random variables using what can be observed (a.k.a Evidence)

**Ex**: Given wet grass \( G = \text{true} \), chance of rain \( R = \text{true} \) is:

\[
P(R = T | G = T) = \frac{P(G = T, R = T)}{P(G = T)} = \frac{\sum_{S \in \{T,F\}} P(G = T, S, R = T)}{\sum_{S,R \in \{T,F\}} P(G = T, S, R)}
\]
Property Graph and Data Management

- Property graph is a data representation model with strong expressiveness.
- Property graph is supported by most graph databases (NoSQL) and also forms the foundation of graph analysis.

  - Vertices
    - Unique ID for each
    - A set of (directed) edges
    - Property: a set of key-value pairs
  
  - Edges
    - Unique ID for each
    - Two end vertices
    - With at least a label
    - Property: a set of key-value pairs
Property Graph Implementation

- **Adjacent list**
  - Similar to CSR, with improvements
  - Utilized by ScaleGraph etc

- **Adjacent matrix**
  - Graph $\rightarrow$ Sparse Matrix
  - Suit to some algorithms (e.g. PageRank),
  - Utilized by IBM GPI

- **Vertex property list + edge property list**
  - Utilized by Spark/GraphX
  - Straightforward and effective data organization
Basic Operators in Property Graph

- **Traversal**
  - **Def:** Visit/Modify vertices following the edges
  - **Implementation:** BFS, DFS,
  - **Application:** SSSP, CF, Loopy Bayesian Inference

- **Graph Editing**
  - **Def:** add/delete/modify vertices, edges, or the property
  - **Implementation:** local update (graphDB), new graphs (Spark/GraphX)
  - **Application:** Finance Surveillance, Hypergraph construction
Graph Systems
Some Existing Products

Visualization
- Gephi
- Graphviz

Analytics
- Boost Graph Library (BGL)
- ScaleGraph

Frameworks
- Dato
- GraphX

Storage
- neo4j
- TITAN
Neo4j System Architecture and Storage Format

- Traversals
- Core API
- Cypher
- Vertex/Edge Cache
- Thread local diffs
- HA
- FS Cache
- i.e. mmap
- Transaction log
- Record files
- Disk

Graph structure and data buffers

Neo’s declarative query language

LFU-protocol

High Availability based on TX

 Transaction log for TX roll-back

Link edges inclined to a vertex using the relationship data structure, imposing some performance issue for handling celebrities in power-law graphs e.g. social network

Easy to implement horizontal partitioning in FS
Titan System Architecture and Storage Format

- Applications
  - TinkerPop Stack: Gremlin, Rextor, Frames, Furnace, & Pipes all rely on Blueprints

Titan

- TitanGraph API
- Blueprints API
- Client API Layer
- Database Layer
- Storage and Index Backend Layer

Storage Backends (at least one is required):
- Cassandra
- HBase
- BerkeleyDB

External Index Backends (optional):
- Elasticsearch
- Lucene

- Store Manager
- Transaction store
- Relations
- Index Store
OrientDB System Architecture

Support distributed platforms, offering key-value store, docDB, and graphDB in one system.
IBM System G Graph Data Organization

- Keeping a chunk of graph data in memory for efficient data retrieval
- On-demand loading loads data only when the vertices and/or edges are accessed
- Stitching graph data together in memory → increase data locality
- Behaving as an in-memory database

Key table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Latest TS</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Latest TS</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Latest TS</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reduce disk access latency

Cache the latest set of pointers in Key table to reduce disk access. Furthermore, cache the Key table in memory when there is enough memory.

Load entire list likely in a single disk access

Contiguously store adjacency edges for vertex K

Edges

Reference counter

Multiversioning

Property
Glance at Graph Computing Engines

Spark/GraphX

Property Graph

Vertex Table (RDD)
Routing Table (RDD)
Edge Table (RDD)

Dato Architecture

GraphLab Create
Scalable Machine Learning

pip install graphlab-create

GraphChi

Compressed In-Core or Out-of-core scalable data structures

(a) Execution interval (vertices 1-2)
(b) Execution interval (vertices 3-4)
(c) Execution interval (vertices 1-2)
(d) Execution interval (vertices 3-4)
Issues within Existing Systems

- Separation of data management and analytics layers results in unnecessary data duplication, adversely hurting the overall performance
  - GraphLab, GraphX —> No data management available
  - Titan —> No clear model for data computing/analytics
- Limited consideration on Scale-up, but relaying on Scale-out for performance improvement, which is inherently different
  - GraphX and Titan cannot use the low-cost sync.
  - Irregularity in graph data access brings high cost to IO, slowing down the overall graph data processing time
Issues within Existing Systems - 2

- **JVM constraints**
  - Productivity and open-source amenable. Java and Scala run on JVMs
  - Irregular data access in graph forms pressure
    - Poor data locality, leads to increased workload in GC
    - E.g. Importing 200M edges into Neo4j on one shot on a server with 1TB results in out of memory issue; Tuning the transaction size in Titan is also quite challenging.
  - JVM abstraction makes it difficult to use low level features, such as NUMA-awareness, GPU devices, etc.

- **Impact by the constraints of RDD**
  - Spark gives up JVM based GC, which may help improve the performance of GraphX; Due to the characteristics of RDD, dynamics graph can result in a lot of data copy, rather than in-place data update

*Every coin has two sides*
Challenges
Challenges

- **Challenges in graph topology**
  - Different types of edges
  - Randomness in graph structure
    - Celebrity nodes in social graph
  - Dynamic change in graph structure

- **Challenges in graph properties**
  - Schema-less in vertices/edges
  - Property type can be arbitrary
    - Simple property => label
    - Complex ones => JSON
  - Some property can be volatile

- **Challenges in infrastructure**
  - Latency can be more sensitive than bandwidth

- **Performance challenges in Graph DB**

---

C. Guestrin, GraphLab Conference 2013.

We process a big graph using 1000 computers

But the graph data fits into a single machine

Do we abuse “Big”?

40M users, 1.2B edges → 34.8 B triangles
Challenges

- Poor data locality results in high IO cost
  - The way how data is stored in memory or disk is not inherently designed for graph
  - Data access patterns in graph computing are highly irregular

- Cost of graph Partition/Sharding
  - Complexity of MinCut can be much higher than some basic graph computing
  - Not really helpful for dynamic graphs
    - ParMETIS takes seconds to hours for partitioning a graph

- Limits of RDBMs on graphs→Native GraphDB
  - Graph is covered by relational model and can be converted
  - Property graph can be represented by tables of vertices, edges and properties
  - Join operation can be the killer
Challenges - Performance

- Understand performance bottleneck by breaking down the execution time
  - Bottleneck comes from the memory sub-system
    - DTLB is inefficient
    - Cache performs well
    - Cache MPKI rate is high

Core graph algorithms from 21 real-world use cases

3 different types of graph computing, with focus on structural traversal, property processing, and graph editing, respectively
Challenges — Input Sensitivity

- **Impact from graph topology**
  - Power-law graph results in imbalanced workload due to dense vertices
    - Dense subgraph, sparse backbone
    - Dense subgraph can be converted into matrices
    - Iterative update in a subgraph
    - Road net is easy to decompose

- **Property type matters**
  - More time spent on property management
  - Computing performance can be negatively impacted

Performance is inconsistent across different graph types
Challenges – Impact of H/W Accelerator

• GPU can be helpful
  • Sufficient acceleration by GPU
  • Requires re-design of the algorithms

• Challenges
  • Data must be transferred to GPU
  • Cost of Host to Device data transfer
  • Difficulty in putting large graph into GPU (Double buffering)
  • Sensitive to input graph data

Speedup of NVIDIA Tesla K40 over 16-core Intel Xeon E5-2670

Memory divergency shows higher sensitivity for graph computing on GPU
Challenges – Scale-out Issue

• Poor data locality and difficult partitioning result in challenges in scaling out the computing
  • Scale-out challenges can be seen in Graph500 analysis
  • Single machine with big memory can help
  • Must be cautious to use many computing nodes

Analysis of data from Graph500

* from Peter Kogge

degraded performance when #core is 100~1000
Breakthrough and Opportunities
Breakthrough on Distributed Graph Traversal Engine

- Traversal is the core operation in graph computing
  - Pretty high throughput achieved
- Variety of techniques are utilized to achieve the goal
  - Efficient scale-free graph partition for parallelization
  - Dynamic workload balance
  - Beamer-based algorithmic innovation
  - Architecture aware optimization

### November 2015

<table>
<thead>
<tr>
<th>No.</th>
<th>Rank</th>
<th>Machine</th>
<th>Installation Site</th>
<th>Number of nodes</th>
<th>Number of cores</th>
<th>Problem scale</th>
<th>GTEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>K computer (Fujitsu - Custom)</td>
<td>RIKEN Advanced Institute for Computational Science (AICS)</td>
<td>82944</td>
<td>663552</td>
<td>40</td>
<td>38621.4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Lawrence Livermore National Laboratory</td>
<td>98304</td>
<td>1572864</td>
<td>41</td>
<td>23751</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>DOE/SC/Argonne National Laboratory Mira (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Argonne National Laboratory</td>
<td>49152</td>
<td>786432</td>
<td>40</td>
<td>14982</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>JUQUEEN (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Forschungszentrum Juelich (FZJ)</td>
<td>16384</td>
<td>262144</td>
<td>38</td>
<td>5848</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Fermi (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>CINECA</td>
<td>8192</td>
<td>131072</td>
<td>37</td>
<td>2567</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Tianhe-2 (MilkyWay-2) (National University of Defense Technology - MPP)</td>
<td>Changsha, China</td>
<td>8192</td>
<td>196608</td>
<td>36</td>
<td>2061.48</td>
</tr>
</tbody>
</table>
Breakthrough on Graph Analytics for Social Media

Social Media Monitoring

Modeling, Tracking and Affecting Information Dissemination in Context
Breakthrough on Graph for Cognitive Computing

Combing graph technology and big data, we provide insights to the data by especially exploring the relationship among various entities. Based on the same dataset and infrastructure, we are able to provide information from 12 different aspects.
Breakthrough on Graph for Anomaly Detection

Use Probabilistic graphical models, we can model the behavior of a complex system, such as the employees in a large enterprise, or a node in a SDN, and detecting possible abnormal behaviors before the real damage occurs.
Opportunities in Graph Technology for Big Data

- **Develop high performance graph computing kernels and primitives**
  - Graph500 technique based architecture-awareness for graph computing
  - Heterogeneous computing and computing near-data technology
- **Reinvent graph technology for supporting cognitive computing**
  - One open platform with multiple graph and graph-related technologies
  - Integral consideration on graphical model, streaming graphs, etc. for AI/IoT
- **Offer vertical solutions to break through separation among technique stacks**
  - Holistic solution for rapidly building industry-level graph analytics solutions
  - Incorporating with market segmentation, such as security, finance, etc.
- **Collaborations and Standardization**
  - Foster collaboration with relevant professional communities to educate the market
  - Developing domain or cross-domain standardizations
Opportunities on Novel Hardware Support to Graph

**GraphBIG@Github**

A group of graph analytics for benchmarking underlying platforms

A simplified IBM System G in-memory graph layer, with similar APIs

Come with performance profiler by taking hardware performance counters, breaking down the execution time into multiple stages to reveal the performance bottleneck

Clone the Code

Code: [https://github.com/graphbig/graphBIG](https://github.com/graphbig/graphBIG)

Doc: [https://github.com/graphbig/GraphBIG-Doc](https://github.com/graphbig/GraphBIG-Doc)

```
-bash:-$ git clone https://github.com/graphbig/graphBIG.git GraphBIG
Cloning into 'GraphBIG'...
remote: Counting objects: 497, done.
remote: Compressing objects: 100% (110/110), done.
remote: Total 497 (delta 57), reused 0 (delta 0), pack-reused 386
Receiving objects: 100% (497/497), 2.07 MiB | 0 bytes/s, done.
Resolving deltas: 100% (229/229), done.
Checking connectivity... done.
-bash:-$
```

Breakdown of Execution Cycles

- **CompStruct**
- **CompProp**
- **CompDyn**

- **Backend**
- **Retiring**
- **BadSpeculation**
- **Frontend**
Opportunities through Community Collaborations

• Co-chair the IEEE Big Data Standardization under BDI
• Co-chair the IEEE Big Data Conference Government & Industry Program in 2016
• Directed the LDBC board, studying graph query standards
• General Vice Chair of HiPC’16
• Program Chair of CBDCom’16

Selected confirmed attendees:
GraphBIG@Github Mini Hands-On
Features of GraphBIG

- **Framework**
  - Based on the property graph framework from real-world graph computing practices

- **Representativeness**
  - Workloads are selected from real-world use cases

- **Coverage**
  - Covers multiple graph computation types, much more than just graph traversal

- **Multicore/GPU**
  - Provides Multicore/GPU workloads under the unified framework

- **Standalone package**
  - Can be compiled without external libraries

- **Profiling tools**
  - Provides tools to profile the code section of interest with hardware performance counters (libpfm code is integrated)

- **Recognition**
  - First comprehensive graph analytics benchmark for architecture research
  - Tech papers announcement in SC’2015 and will be on VLDB’2016
### Graph Analytics Benchmark

<table>
<thead>
<tr>
<th>Rich analytics available</th>
<th>Each analytics may have multiple algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>bench_BFS</td>
<td>Makefile</td>
</tr>
<tr>
<td>bench_DFS</td>
<td>bfs_data_thread_centric.cu</td>
</tr>
<tr>
<td>bench_TopoMorph</td>
<td>bfs_data_warp_centric.cu</td>
</tr>
<tr>
<td>bench_betweennessCentr</td>
<td>bfs_topo_atomic.cu</td>
</tr>
<tr>
<td>bench_connectedComp</td>
<td>bfs_topo_frontier.cu</td>
</tr>
<tr>
<td>bench_degreeCentr</td>
<td>bfs_topo_thread_centric.cu</td>
</tr>
<tr>
<td>bench_gibbsInference</td>
<td>bfs_topo_unroll.cu</td>
</tr>
<tr>
<td>bench_graphColoring</td>
<td>main.cpp</td>
</tr>
<tr>
<td>bench_graphConstruct</td>
<td>ref_output</td>
</tr>
<tr>
<td>bench_graphUpdate</td>
<td></td>
</tr>
</tbody>
</table>

- bench_kCore
- bench_pageRank
- bench_shortestPath
- bench_triangleCount
- ubench_add
- ubench_delete
- ubench_find
- ubench_traverse

- cudalib
- gpu_BFS
- gpu_BetweennessCentr
- gpu_ConnectedComp
- gpu_DegreeCentr
- gpu_GraphColoring
- gpu_SSSP
- gpu_TriangleCount
- gpu_kCore
Mini Hands-on: Clone the Source

Fetch Code

- Code: https://github.com/graphbig/graphBIG
- Doc: https://github.com/graphbig/GraphBIG-Doc
Mini Hands-on: Compile

Compile

- Require: gcc/g++ (>4.3), gnu make
- Just “make all”
Mini Hands-on: Test Run

Test Run

- Just “make run”
- Using default “small” dataset
Mini Hands-on: User-Defined Analytics

typedef openG::extGraph<vertex_property, edge_property> graph_t;
typedef graph_t::vertex_iterator vertex_iterator;
typedef graph_t::edge_iterator edge_iterator;

void graph_traverse(graph_t& g)
{
    vertex_iterator vit;
    uint64_t vcnt=0;
    uint64_t ecnt=0;
    for (vit=g.vertices_begin(); vit!=g.vertices_end(); vit++)
    {
        vit->set_property(vertex_property(vcnt++));
        edge_iterator eit;
        for (eit=vit->out_edges_begin(); eit!=vit->out_edges_end(); eit++)
        {
            eit->set_property(edge_property(ecnt++));
        }
    }
}
Mini Hands-on: Dataset Download

More Datasets

- Download: https://github.com/graphbig/graphBIG/wiki/GraphBIG-Dataset
- Untar and specify the correct path in benchmark argument “--dataset”
- Other 3rd party datasets (.csv format) are also possible
THANK YOU

Yinglong Xia
yinglong.xia.2010@ieee.org