Large Scale Graph Processing Opportunities and Challenges, an Industry Perspective

Oracle Labs
Hassan Chafi
Sr. Director, Research and Advanced Development
Safe Harbor Statement

The following is intended to outline our research activities and general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Graph Analysis Applications from Industry
Graph-Based Approach: Basic Idea

• Represent your dataset as a graph
  – Entities become vertices
  – Relationships become edges

• For the purpose of
  – Data modeling
  – Data analysis
Graph Benefits

• For Data Modeling
  – Intuitive global view on the data
  – Easier and faster query
  – Multi-domain (master) data management

• For Data Analysis
  – New insights about the dataset
  – Less turn around time
  – Increased quality of answer
Example: Network Management for a Cloud Provider

- Cloud management using Graph
  - Configuration: machines and their interconnections
    - Racks, machines, switches, ports, wires ...
  - Quickly (and conveniently) look up configurations

Count the number of Machines (indirectly) connected to host “sea2” by Location

```
SELECT z.name, count(*)
WHERE (x WITH name = 'sea2-host1') -/:connects_to*/- (y: Machine),
  (y) <- [:belongs_to]- (z: Location)
GROUP BY z
```

Looking for multi-hop connections

Location: South Dakota

Graph model makes it easy to write a query and fast to get answer.
Example: AML (Anti-Money Laundering)

- Graph analysis works as a key step in AML Solutions
  - A rule-based system evaluates each transaction and creates alerts on suspicious ones
  ➔ This creates a graph of alerts
  - Detect cases where many correlated alerts around small number of entities.

Detection through graph analysis: pattern matching + connected component
Example: Topic analysis in an Online Forum

- **Analysis Goals:**
  - Identify popular *topics* in on-line forum
  - Understand how these topics evolve
  - Detect expert users in certain topics

- **Graph Approach**
  - Create graph from postings and tags
  - Apply graph partitioning (*community detection*) algorithms

Comparing to traditional ML approach (e.g. LDA), this approach often results better quality of answer, with less susceptibility to hyper-parameters.
Example: Fake Account Detection in SNS

- Analyze message streams in Social Network for marketing purpose
  ➔ However, a lot of noisy messages are generated by Fake accounts (bots)
- Applying graph analysis
  - Create a graph view of messages and SNS accounts
  - Detect accounts that have different neighborhood communication patterns
  - Propagate *tainted* account over the graph to identify further fake accounts

- More than 16% of accounts were labelled as bots
- Eliminating messages from bots resulted significant changes in trend ranking
Example: Electricity Network Risk Analysis

- Electric Power Distribution Network as a graph
  - Analyze importance of nodes in the network
  - ➔ What nodes to pay attention during emergency (e.g. storm season)

Failure of this node has higher impact than others
Combining Graph Analysis and Machine Learning

• Graph analysis can augment Machine Learning
  – Typical machine learning techniques train models based on directly observed features
  – Graph analysis can provide additional *strong* signals by analyzing relationships
  – Which make ML model more accurate

<table>
<thead>
<tr>
<th>Feature1</th>
<th>Feature 2</th>
<th>Feature 3</th>
<th>Feature 4</th>
<th>Feature 5</th>
<th>Feature 6</th>
<th>Feature 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predictive Model
Example – Inferring department

- Dataset: students taking courses in a university

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student department</th>
<th>Student type</th>
<th>Student Age</th>
<th>Student Gender</th>
<th>... Other Features (Student)</th>
<th>Course Name</th>
<th>Course department</th>
<th>Course Type</th>
<th>Course Credit</th>
<th>... Other Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>ME</td>
<td>Transfer</td>
<td>32</td>
<td>F</td>
<td>........</td>
<td>Numeric Methods</td>
<td>??</td>
<td>Lecture</td>
<td>4</td>
<td>........</td>
</tr>
<tr>
<td>Bob</td>
<td>??</td>
<td>Local</td>
<td>23</td>
<td>M</td>
<td>........</td>
<td>Numeric Methods</td>
<td>??</td>
<td>Lecture</td>
<td>4</td>
<td>........</td>
</tr>
<tr>
<td>Bob</td>
<td>??</td>
<td>Local</td>
<td>23</td>
<td>M</td>
<td>........</td>
<td>Machine Learning</td>
<td>CS</td>
<td>Lecture</td>
<td>3</td>
<td>........</td>
</tr>
<tr>
<td>Chris</td>
<td>??</td>
<td>Foreign</td>
<td>24</td>
<td>M</td>
<td>........</td>
<td>Machine Learning</td>
<td>CS</td>
<td>Lecture</td>
<td>3</td>
<td>........</td>
</tr>
</tbody>
</table>

- How to infer missing departments information?
Possible Approaches

(#1) Create CNN model directly from attributes

(#2) Create Graph to infer dept from topology

(#3) Apply text processing techniques

(#4) Combination of these

Possible Approaches

(#1) Create CNN model directly from attributes

(#2) Create Graph to infer dept from topology

(#3) Apply text processing techniques

(#4) Combination of these

[Numeral Methods] This course offers an advanced introduction to numerical linear algebra. Topics include direct and iterative methods for linear systems, eigenvalue decompositions and QR/SVD factorizations, stability and accuracy of numerical algorithms, the IEEE floating point standard, sparse and structured matrices, preconditioning, linear algebra software. Problem sets require some knowledge of MATLAB®.
Results

<table>
<thead>
<tr>
<th>Approach</th>
<th>Achieved Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#1) Direct features</td>
<td>57.3%</td>
</tr>
<tr>
<td>(#2) Graph Topology</td>
<td>71.7%</td>
</tr>
<tr>
<td>(#3) Text Processing</td>
<td>73.6%</td>
</tr>
<tr>
<td>(#4) Combined Approach</td>
<td>78.1%</td>
</tr>
</tbody>
</table>

- Train on direct features do not work so well
- For graph topology, you can skip training
- Text processing uses most relevant information
- Combining (#2) + (#3) gave best result
Industry Perspective: Challenge is Tooling

• Applying graph analysis is still challenging for customers
• Available solutions still not satisfactory
  – Lack of integration
  – Lack of flexibility
  – Lack of scalability
  – Lack of performance
Introduction to PGX
Our Approach (1): Integrated Solution

- An end-to-end solution to users via multiple layers of integrations

Multiple language bindings for remote execution via REST

In-memory Engines for (Single Machine or Distributed) Execution

Works well with open-source frameworks e.g. Apache Spark

Integrated with various Oracle data storage technology
Combining Modeling and Analysis

**Graph Database**
- Two different data models
  - RDF
    - John
    - bornIn: NewYork
    - hasCar: Hyundai
    - friendOf: Mary
    - LiveIn: NewYork
    - favor: poetry
  - Property Graph
    - name: John
      - hasCar: Hyundai
      - bornIn: NewYork
    - name: Mary
      - liveIn: NewYork
      - favor: poetry

**Graph Framework**
- Execution environments
  - Single Machine
    - Virtuoso
    - PGX.SM
    - igraph
    - NetworkX
  - Cluster Environment
    - GraphX
    - GraphLab
    - PGX.DIST

**Focus**: data modeling, management, and querying

**Focus**: graph traversal and computation

Support both usages in one environment
Our Approach (2): Flexibility via DSLs

Concurrent remote clients calling PGX API via REST

Further flexibility (and performance) via two DSLs

PGX client can be Notebook or Visualizer

DSLs for Graph Query or Custom Algorithm

Rest API

PGX (Runtime)

Client
Groovy, R, JavaScript, …
PGQL – DSL for pattern-matching graph query

• About PGQL
  – A query language for property graph
  – No standard yet
  – Cypher from Neo4J (proprietary) has several issues both in syntax and semantics
  – PGQL is our own proposal, implemented in PGX

• Open Specification
  – Academic Paper: GRADES 2016
  – Open-sourced the specification to the public (pgql-lang.org)
  – Open-source Parser on GitHub
A Quick Look at PGQL

• Example Query

PATH eoae := ()-[[:enemy]->()]->[[:enemy]->()]  
SELECT z.name, y.name, count(*)  
FROM snGraph  
WHERE  
  (z:Person WITH name = 'Amber') -/:eoae*/-> (y),  
  (x:Person WITH city = 'NY') <->[e:friend] (y),  
  x.age > y.age  
GROUP BY z, y

// for each Amber (z)  
// find every her (enemy-of-enemy) y  
// get number of x, friend of y in NY, older than y

SQL-like syntax:  
SELECT/FROM/WHERE/GROUP BY ...

Graph Pattern Matching:  
(VERTEX:label)-[EDGE:label]->() 

Recursive Path Query:  
(x) -/:pattern*/->(y)
PGQL Path-Query Implementation

- Recursive Path Queries
  - **Brute-Force**: Build FSM and evaluate it while searching
  - **Reachability Graph (RG)**:
    - For each vertex, find all vertices reachable by one “unit” of recursion
    - BFS on the induced graph

→ Falls back to Brute-force if there is not enough memory to build RG index
Green-Marl: DSL for Graph Algorithm

• Approach
  – Implement custom algorithm with DSL
  – Compiled and executed at server-side

• Benefits
  – No network overhead (vs. remote access)
  – Use internal data structure directly (vs. API overhead)
  – Parallelization and Optimization from compiler
Network and API Overhead

Significant (orders of magnitude) performance benefits by avoiding Network and API

<table>
<thead>
<tr>
<th>Runtime in seconds (log scale)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include Network</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Compilation time

Epinions | LiveJournal

PGX

DSL Compiler

Compiled Program

API

Internal Data-Structures

DSL Code

Network (Network)
Compiler Optimization

- Example Code: Closeness Centrality

```java
proc closeness(G: graph; CC: nodeProp<double>)
{
    // start from every vertex in the graph
    foreach (s : G.nodes) {
        long foundNodes = 0;
        long levelSum;

        // do BFS and sum up distance to other vertices
        inBFS(v: G.nodes from s) {
            foundNodes++;
            levelSum += currentBFSLevel();
        }

        // centrality is reciprocal of distance sum
        s.CC = 1.0 / (double) levelSum;
    }
}
```

- Apply compiler optimization
  - Auto apply MS-BFS technique
  - Save user from writing complicated implementation code
Optimization Result

• Performance Improvement
  – multiple orders (vs. not-applying optimization)
• User convenience
  – Algorithm code becomes performance code transparently
  – Compiler takes care implementation details: context management, memory consumption, ...
• Reusability
  – Same optimization applies to other algorithms with similar pattern
  ➔ Betweenness centrality, diameter, periphery, ...

<table>
<thead>
<tr>
<th>Graph Instance</th>
<th># vertices</th>
<th># edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epinions</td>
<td>75,879</td>
<td>508,837</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>4,848,571</td>
<td>68,993,773</td>
</tr>
<tr>
<td>Twitter</td>
<td>41,652,230</td>
<td>1,468,365,182</td>
</tr>
</tbody>
</table>
Our Approach (3) : Distributed Execution

• Distributed Runtime of PGX
  – Capable of loading large graphs in-memory
  – The graph is partitioned and loaded into multiple machines
  – Graph algorithms are run in a distributed way using remote communication

• Goal to support the same API as PGX
  – Currently supports a subset
  – A PGX Client can connect to PGX.DIST server in the same way as to PGX.SM server
Techniques for PGX.DIST

Balancing workloads and avoiding communication

Basic Ideas
- Edge Partitioning
- High Degree Node Replication

Additional Ideas
- Edge Chunking
- High Degree Node Privatization

Implementing very efficient communication mechanism

Basic Ideas
- Latency Hiding in Remote Access
- Remote Data Pulling
- Sending computation

Additional Ideas
- Run-to-Completion Based Task system
- Companion Buffer

Essentially, we combined conventional techniques with our own ideas.

Detailed information can be found in our papers and patents.
DSL Algorithm Compilation for PGX.D

- PGX.D (internal) API is complicated
  - Designed for execution performance
  - Let compiler generate code from DSL

```
proc pagerank_like (g: graph) {
  nodeProp<double> rank;
  nodeProp<double> in_sum;
  foreach (n : g.nodes)
    foreach (m : n.outNbrs)
      n.in_sum += m.rank / m.outDegree();
}
```
DSL Algorithm Compilation for PGX.D

- PGX.D (internal) API is complicated
  - Designed for execution performance
  - \( \rightarrow \) Let compiler generate code from DSL

```cpp
proc pagerank_like (g: graph) {
  nodeProp<double> rank;
  nodeProp<double> in_sum;

  foreach (n : g.nodes)
    foreach (m : n.outNbrs)
      n.in_sum += m.rank / m.outDegree();
}
```

```cpp
auto job0_src = SOURCE_FUNCTION(CAPTURE_LIST(&),
  data_access, local_vertex_id, local_edge_id, bytes_available,
  buffer_ptr, write_ctx, is_intermachine,
  FUNCTION_BODY(
    message0 *buffer = (message0*) buffer_ptr;
    buffer->g_pg_rank = data_access->template
    get_nprop<double>(g_pg_rank, local_vertex_id);
    buffer->numOutNbrs = data_access->
    get_num_original_out_edges(local_vertex_id);
    return sizeof(message0);
  ));

auto job0_tgt = TARGET_FUNCTION(CAPTURE_LIST(&),
  data_access, local_vertex_id, local_edge_id, buffer_ptr,
  is_intermachine,
  FUNCTION_BODY(
    message0 *buffer = (message0*) buffer_ptr;
    data_access->template set_nprop<double, PGX_SUM>(prop_in_sum0,
      local_vertex_id,
      (buffer->g_pg_rank) / ((double)
       (buffer->numOutNbrs)));
    return sizeof(message0);
  ));
```

\( \text{std::shared_ptr<pgx_job> job0 = sys}\)

\( \text{comm_thread_group_id;} \)

\( \text{job0->transfer_data(job0_src, job0_tgt)} \)
Distributed Pattern Matching

- Implementing pattern-matching for PGX.DIST
  - The bulk-synchronous engine would not be suitable
  - Need too much synchronization and partial solutions

- Approach:
  - Implement Asynchronous traversal engine
  - Construct and bring match-context during async traverse
Distributed Pattern Matching

• Key challenges in Implementation
  – Concurrency control: messaging should not block overall progress
  – Flow control: avoid flooding and deadlock
    • Bounded buffer control
  – Termination detection: ensure and detect termination condition

• Current status
  – Initial version working (GRADES 2017)
  – Runs PGQL
  – Scales well for large query
  – Not too bad even for small query (compared to single machine)
Performance Overview
Analytics Performance: vs. Apache Spark (GraphX)

Several orders of magnitude difference in performance

Hardware: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz - 256 RAM
Network: Infiniband (40Gbps)
Benchmark Results from Academia

• Graphalytics (VLDB2016)
  – A benchmark framework for graph analysis
  – composed of multiple algorithms and graphs
  – Joint work among academia and industry
    • Industry: IBM, Intel, SAP, Huawei
    • Academia: Led by TU Delft

• Note: PGX is a commercialized end-to-end product
  – Much richer feature sets than other graph engines.

• Outperforms Giraph and GraphX by far
  • Top-tier performance and scalability
Graph Query Performance: vs. Neo4j

Dataset: “RDBMS Code as Graph” : 5M nodes and 64M edges

Numbers shown are “hot” numbers: Neo4j is no longer accessing disk.

Hardware: 88 x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz - 256 RAM
Warmup: ignore the first two runs, measure the third run
Index usage: no special indexes (use defaults)
Graph Query Performance: vs. Apache Spark (GraphFrames)

PGX is up to 1600x faster

Execution Time (ms)

Data set: LUBM3, Spark2.1.0, GraphFrames 0.5.0 vs. PGX2.5.0
Measured on X5-2 machines
Summary and Closing Thoughts

• Graph processing is finding use cases in industry, but still early days

• Oracle is building both specialized systems (good/fast for learning and papers) but also adding Graph capabilities to mature products like the database (slow/complex to achieve but highest payoff)

• Graph is part of a set of emerging data analytics tools (including data IDEs/Studios), interesting areas of research when one looks at the whole analytical pipeline
  – In particular, need easy to use (nearly automated) graph creation tools