Overview of Oracle Big Data Spatial and Graph Property Graph

Zhe Wu, Ph.D. Architect

Oracle Spatial and Graph
Jan, 2016
The following is intended to outline our 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.
The Property Graph Data Model

- **A set of vertices (or nodes)**
  - each vertex has a unique identifier.
  - each vertex has a set of in/out edges.
  - each vertex has a collection of **key-value** properties.

- **A set of edges (or links)**
  - each edge has a unique identifier.
  - each edge has a head/tail vertex.
  - each edge has a label denoting type of relationship between two vertices.
  - each edge has a collection of **key-value** properties.

https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model
Architecture of Property Graph Support on Big Data Platform

Graph Analytics
- Parallel In-Memory Graph Analytics (PGX)

Graph Data Access Layer (DAL)
- Blueprints & Lucene/SolrCloud

Scalable and Persistent Storage Management
- Oracle NoSQL Database
- Apache HBase

Java APIs

REST/Web Service

Property Graph formats
- GraphML
- GML
- Graph-SON
- Flat Files

Java, Groovy, Python, ...

RDF (RDF/XML, N-Triples, N-Quads, TriG, N3, JSON)
Common Graph Analysis Use Cases

Product Recommendation

- Recommend the most similar item purchased by similar people

Influencer Identification

- Find out people that are central in the given network – e.g. influencer marketing

Community Detection

- Identify group of people that are close to each other – e.g. target group marketing

Graph Pattern Matching

- Find out all the sets of entities that match to the given pattern – e.g. fraud detection
Property Graph Workflow

• Graph Data Management
  – Raw business data is converted to a graph schema
  – In Database graph queries using SQL (useful for breath first search)

• Analysis and Exploration (in-memory analysis engine)
  – Data scientists try different ideas (algorithms) on the data
  – Flexible, interactive, iterative, small-scale (sampled), …. 
Key Features: Efficient Blueprints API Implementation

• OracleElement
  – getId/setProperty/getProperty/getPropertyKeys/removeProperty

• OraclePropertyGraph
  – addVertex/getVertex/getVertices/removeVertex
  – addEdge/getEdge/getEdges/removeEdge

• OracleVertex
  – addEdge/getVertices/getEdges

• OracleEdge
  – getLabel/getVertex
Key Features: Parallel Property Graph Data Loader

• Splitter threads
  – split vertex/edge flat files into multiple chunks

• Loader threads
  – load each chunk into database using separate database connection

• Use Java pipes to connect Splitter and Loader threads
  – Splitter: PipedOutputStream
  – Loader: PipedInputStream

• Example
  
  opgdl = OraclePropertyGraphDataLoader.getInstance();
  vfile = "/home/alwu/pg-bda-nosql/demo/connections.vertices";
  efile = "/home/alwu/pg-bda-nosql/demo/connections.edges";
  opgdl.loadData(opg, vfile, efile, 48);
Key Features: Multiple Property Graph Data Format Support

• GML, GraphML, GraphSON
• Oracle-defined Property Graph flat files
  – Vertex file, Edge file
  – Support basic data types + Date with Timezone + Serializable objects
  – Allow multiple data types to be associated with one key
  – UTF8 based
  – Example
    1,name,1,Barack%20Obama,,
    1,age,2,,53,
    1,likes,1,scrabble,,
    2,likes,5,,,2009-01-20T00:00:00.000-05:00
    2,occupation,1,44th%20president%20of%20United%20States%20of%20America,,, 
Partitioned Parallel Data Loader (v1.1)

- Parallel Data Loader
  - Split vertex/edge flat filesstreams into multiple chunks
  - Load each chunk into database using a separate database connection

- Enhancements (new API)

```
loadData(opg /*OraclePropertyGraph instance*/,
    saVertexFlatFileNames /* an array of vertex flat files */,
    saEdgeFlatFileNames    /* an array of edge flat files */,
    lVertexSrcOffsetlines /* line offset of vertices */,
    lEdgeSrcOffsetlines /* line offset of edges */,
    lVertexSrcMaxlines /* maximum lines of vertices to be loaded */,
    lEdgeSrcMaxlines /* maximum lines of edges to be loaded */,
    iDop /* degree of parallelism */,
    iPartitions /* number of partitions */,
    iOffset /* offset of partitions */,
    dll /* DataLoaderListener instance to report progress and control loading process when errors occur */)
```
Partitioned Parallel Data Loader (v1.1)

• Example: load people graph

```java
KVStoreConfig kconfig = new KVStoreConfig(szStoreName, hhosts);
OraclePropertyGraph people = OraclePropertyGraphgetInstance(kconfig, "people");
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
DataLoaderListener dll = SimpleLogBasedDataLoaderListenerImpl.getInstance(10 /* frequency */,
false /* continue on error */);
String[] opvFiles = {"../data/people1.opv", "/data/people2.opv"};
String[] opeFiles = {"../data/people1.ope", "../data/people2.ope"};
opgdl.loadData(people,
    opvFiles, opeFiles,
    0, 0,
    -1, -1,
    36 /* dop */,
    1 /* total partitions *//, 0 /* partition ID *//, dll)
```
Sub-graph Extraction using Filter Callbacks (v1.1)

• Motivation
  – Users only interested in certain sub-graphs, e.g., employees in a specific cost center
  – Speedup graph scan

• Users need to implement interfaces VertexFilterCallback/EdgeFilterCallback:
  – boolean keepVertex(OracleVertexBase v)/boolean keepEdge(OracleEdgeBase e)

• Users can use these filter callbacks by setting the default Filter Callback in an OraclePropertyGraph instance
  – setDefaultVertexFilterCallback(vfc)/setDefaultEdgeFilterCallback(efc)
  – getVertexFilterCallback()/getEdgeFilterCallback()

• Or explicitly passing the parameter into the getVertices/getEdges API call
  – getVertices(..., vfc, ...)/getEdges(..., efc, ...)
  – getVerticesPartitioned(..., vfc, ...)/getEdgesPartitioned(..., efc, ...)
Sub-graph Extraction using Filter Callbacks (v1.1)

• Example: Retrieve only Oracle employees in SM01 cost center

```java
class PeopleVertexFilterCallback implements VertexFilterCallback {
    PeopleVertexFilterCallback() {
    }

    public boolean keepVertex(OracleVertexBase v) {
        String cc = v.getProperty("cc");
        if (cc != null && cc.equals("SM01")) {
            return true;
        } else {
            return false;
        }
    }
}
```

```java
VertexFilterCallback vfc = new PeopleVertexFilterCallback();
people.setDefaultValueFilterCallback(vfc);
people.getVertices();
```

OR

```java
VertexFilterCallback vfc = new PeopleVertexFilterCallback();
people.getVertices(keys /* keys, can be null */, vfc);
```
Text Search through Apache Lucene/Solr Cloud

- Integration with Apache Lucene/Solr
- Support manual and auto indexing of Graph elements
  - Manual index:
    - `oraclePropertyGraph.createIndex("my_index", Vertex.class);`
    - `indexVertices = oraclePropertyGraph.getIndex("my_index", Vertex.class);`
    - `indexVertices.put("key", "value", myVertex);`
  - Auto Index
    - `oraclePropertyGraph.createKeyIndex("name", Edge.class);`
    - `oraclePropertyGraph.getEdges("name", "*hello*world");`
  - Enables queries to use syntax like "*oracle* or *graph*"
SolrCloud: Fuzzy and Faceted Search of Graph Elements

Facet: "role"

Search vertices located in the "States" (string type)

http://www.slideshare.net/thelabdude/solr-exchange-introtosolrcloud
Parallel Text Query using Text Indexes (v1.1)

Application code

Index Connection 1

Index Connection 2

Index Connection ...

Index Connection N

Graph Data Access Layer (DAL)

Index Iterable 1

Index Iterable 2

Index Iterable ...

Index Iterable N

OracleIndex.getPartitioned

Scalable and Persistent Storage Management

Oracle NoSQL Database

Apache HBase

Apache Lucene

Sub Directory 1

Sub Directory 2

Sub Directory 3

Sub Directory 4

Sub Directory 5

Sub Directory 6

Sub Directory ...

Sub Directory N

Apache SolrCloud

SolrCloud Node 1

SolrCloud Node 2

SolrCloud Node ...

SolrCloud Node N

SolrCloud Node 1

SolrCloud Node 2

SolrCloud Node ...

SolrCloud Node N

OracleIndex.getPartitioned
Parallel Text Query APIs (v1.1)

- `getPartitioned(SearcherManager[] conns, String key, Object value, boolean useWildcards, int startSubDirID)`
  - Each connection in the array needs to be linked to the appropriate sub-directory, as each sub-directory is independent from the rest.
  - For each sub-directory in `[startSubDirID, startSubDirID + conns.length)`
    - Execute the text query (to get all elements matching the given Key/Value pair) using the sub-directory SearcherManager from the connections array
    - Each query will produce a single `Iterable<Element>` object with all the matches found in the sub-directory.

- `getPartitioned(CloudSolrServer[] conns, String key, Object value, boolean useWildcards, int startShardID)`
  - Each connection in the array is an HTTP connection linked to the client endpoint of the zookeeper quorum containing the cloud state.
  - For each shard in `[startShardID, startShardID + conns.length)`
    - Execute the text query (to get all elements matching the given Key/Value pair) using a CloudSolrServer object from the connections array
    - Each query will produce a single `Iterable<Element>` object with all the matches found in the shard
## Getting Property Graph Metadata (v1.1)

### Graph Names

- **OraclePropertyGraphUtils.getGraphNames(dbArgs)**
  List all the graphs stored in the back-end database. This requires executing a table metadata function to look up for the existing vertices/edge tables.

### Total # of vertices/edges

- **opg.countVertices(), opg.countEdges(), opg.countVertices(dop), opg.countEdges(dop)**
  Get the minimum/maximum element ID found in the graph. Uses parallel scans on the vertex/edge tables and skips vertices/edges object creation.

### Min/Max ID for vertices/edges

- **opg.getMinVertexID(), opg.getMaxVertexID(), opg.getMinEdgeID(), opg.getMaxEdgeID()**
  Get the minimum/maximum element ID found in the graph. Uses parallel scans to the vertex/edge tables and skips vertices/edges object creation.

### Property Names for vertices/edges

- **opg.getVertexPropertyNames(dop, timeout, propertyNames), opg.getEdgePropertyNames(...)**
  Get the list of property names used in at least one vertex (edge) of the property graph. Uses parallel scans to read all vertices/edges and filter out property names.

### Table splits

- **opg.getVertexTableSplits(), opg.getEdgeTableSplits()**
  Gets the number of splits used to distribute the vertices (edges) in the graph tables. The number of splits is deeply related to parallel scan operations.
Connecting to Secure NoSQL DB (v1.1)

• Approach 1:
  – Set `oracle.kv.security` to the login properties file
    `-Doracle.kv.security=/<your-path>/login_properties.txt`, or
    `setenv JAVA_OPTIONS "-Doracle.kv.security=/<your-path>/login_properties.txt"
  – Call `OraclePropertyGraph.getInstance(kconfig, szGraphName)`

• Approach 2 (new Java API in v1.1):
  – Call `OraclePropertyGraph.getInstance(kconfig, szGraphName, username, password, truStoreFile)`
    username: myusername
    password: the credential set when creating user myusername
    truStoreFile: the path to the client side trust store file client.trust

No code changes for existing application!
Connecting to a Secure Apache HBase (v1.1)

• Using an HBase-site.xml and Java Authentication and Authorization Service (JAAS) configuration file to get the credentials information:

```java
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    useTicketCache=true
    keyTab="/path/to/your/keytab/user.keytab"
    principal="your-user/your.fully.qualified.domain.name@YOUR.REALM";
};
```

```bash
java -Djava.security.auth.login.config=/path/jaas.conf -Djava.security.krb5.conf=/etc/krb5.conf -classpath ./classes/:../../lib/*:/path-to-hbase-conf <DALJavaApplication>
```

No code changes for existing application!
Key Features: In-Memory Analyst (1)

• PGX: an in-memory, parallel framework for fast graph analytics
• Property Graph for Oracle NoSQL Database and PGX integration
  – Can load a graph (through data access layer) from Oracle NoSQL Database
  – PGX handles analytic workloads while the data access layer handles transactional workloads
  – PGX supports automatic refresh from an Oracle Database via delta-update
Key Features: In-Memory Analyst (2)

• Analytics API examples

```java
// get an in-memory analyst
session=Pgx.createSession("session_ID_1");
analyst=session.createAnalyst();

// Read the graph from database into memory
pgxGraph = session.readGraphWithProperties(opg.getConfig());

// perform analytical functions
countTriangles()
shortestPathDijkstra(...)
pagerank(....)
wcc()
communitiesLabelPropagation
```
Built-in Algorithms and Graph Mutation

- A rich set of built-in (parallel) graph algorithms

  - Detecting Components and Communities
    - Tarjan’s, Kosaraju’s, Weakly Connected Components, Label Propagation (w/ variants), Spasification

  - Ranking and Walking
    - Pagerank, Personalized Pagerank, Betweenness Centrality (w/ variants), Closeness Centrality, Degree Centrality, Eigenvector Centrality, HITS, Random walking and sampling (w/ variants)

  - Evaluating Community Structures
    - Conductance, Modularity, Clustering Coefficient (Triangle Counting)

  - Path-Finding
    - Hop-Distance (BFS), Dijkstra’s, Bi-directional Dijkstra’s, Bellman-Ford’s

  - Link Prediction
    - SALSA (Twitter’s Who-to-follow)

  - Other Classics
    - Vertex Cover

- as well as parallel graph mutation operations

  - Create Bipartite Graph
  - Create Undirected Graph
  - Sort-By-Degree (Renumbering)
  - Filtered Subgraph
  - Simplify Graph

  The original graph

  Filter Expression

  Left Set: “a,b,e”
Key Features: Flexible Graph Analytics Deployment

• Embedded
  – Java application embeds the in memory analytics component
  – Same address space
  – Efficient but requires sufficient RAM on the client side

• Remote
  – Graph analytics deployed in a web container
    • WebLogic Server, Tomcat, Jetty
    • SSL/HTTP Basic Authentication

  – Yarn Integration
    • Let Yarn find the necessary CPU/RAM to deploy the in memory graph analytics
Example REST Interface

• Installation
  – Get the following three jar files under `/opt/oracle/oracle-spatial-graph/property_graph/dal/webapp`
    • rexster-server-2.3.0.jar, rexster-core-2.3.0.jar, and rexster-protocol-2.3.0.jar, grizzly*.jar
  – Configure rexster-opg.xml
  – `../rexster-opg.sh --start -c rexster-opg.xml`

• Usage examples
  setenv pg_graph http://bigdatalite.us.oracle.com:8182/graphs/connections
  curl -X POST "${pg_graph}/vertices/1001"
  curl -X POST "${pg_graph}/vertices/1002"
  curl -X POST "${pg_graph}/edges/2000?_outV=1001&_label=admires&_inV=1002"
  curl -X DELETE "${pg_graph}/edges/2000"
Example Python Interface

• Installation
  – property_graph/examples/pyopg/README

• Usage
  – cd /opt/oracle/oracle-spatial-graph/property_graph/examples/pyopg ./pyopg.sh
  ipython notebook

```python
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(16,12));
community_frame["size"].plot(kind="bar", title="Communities and Sizes")
ax.set_xticklabels(community_frame.index);
```
Tools & Integration
Support Big Data SQL

Oracle RDBMS

External table

Apache Hive

SQL based aggregation and analytics

Apache HBase

Bring the "Connections" property graph data into Oracle RDBMS via external table
Support of Cytoscape Visualization
Support of Cytoscape Visualization (2)
Support of Cytoscape Visualization (3)
Support of Cytoscape Visualization (4)
Integration with Tom Sawyer Perspectives via property graph REST APIs
Integration with Tom Sawyer Perspectives (2) via property graph REST APIs
Integration with Tom Sawyer Perspectives (3) via property graph REST APIs
Performance
Linear Performance Scalability: Data Loading on NoSQL
(2 Clients, 48G Heap, DOP 36)

- 3M edges
- 70M edges
- 1.5B edges
- 3B edges

(a) NoSQL 8-Node BDA: Oracle NoSQL Database Enterprise Edition 3.2.5, 8 Storage Nodes, 12 Replication Nodes/Storage Node, 8.5G Heap Size/Replication Node, Replication Factor 1
Client: Heap Size 48G, 2 Clients

(b) NoSQL 9-Node Clutter: Oracle NoSQL Database Enterprise Edition 3.2.5, 9 Storage Nodes, 4 Replication Nodes/Storage Node, 31G Heap Size/Replication Node, Replication Factor 1
Client: Heap Size 48G, 2 Clients
Performance on In-Memory Analyst
LiveJ (70M edges, 1 Client, 48G Heap, 24 regions, 2 splits per region)

Count triangles

Page Ranking

a) BDA Cluster: Enterprise Cloudera CDH 5.4 +Apache HBase 1.0.0, 4 Region Server nodes, 1 co-located Master node (48GB per node, Processors 24). Heap Size 31G, -Xmn256m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70 -XX:+UseCMSInitiatingOccupancyOnly.

b) BDA Cluster: Enterprise Cloudera CDH 5.4 +Apache HBase 1.0.0, 8 Region Server nodes, 1 co-located Master node (126 GB per node, Processors 72). Heap Size 31G, -Xmn256m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70 -XX:+UseCMSInitiatingOccupancyOnly.
Performance of Basic Operations against Twitter Graph
(50K vertices, 50K edges, 10 K/V pairs, 48G Heap)

NoSQL 9-Node Cluster: Oracle NoSQL Database Enterprise Edition 3.2.5, 9 Storage Nodes, 4 Replication Nodes/Storage Node,