Graph Analytics with Greenplum and Apache MADlib

Pivotal Korea

HongDon Lee, Sr. Data Scientist (holee@pivotal.io)

30th Jan. 2019
Agenda

1. Why Graph Analytics?
2. What is Graph Analytics?
3. Graph Analytics w/ MADlib
“Nothing ever exists entirely alone. Everything is in relation to everything else (緣起)”

“Learn how to see: Everything is connected to everything else”

“In nature we never see anything isolated, but everything in connection with something else which is before it, beside it, under it and over it”

Buddha

Leonardo da Vinci

Goethe
What a Small World!

“6 Degrees of Separation”

1973, Stanley Milgram, Small-world experiment
From Reductionism to Holism

**Reductionism**

“Divide and Conquer”

**Holism**

“Everything has to be understood in relation to the whole”
From Individual to Relation

Features
- Demographics
- Behaviors
- Preferences
- Economic Status
- Education Background

At the individual level

Cross-sectional Perspective

Longitudinal Perspective

Who are you?

Time

2019.01.01
2019.01.02
2019.01.03
2019.01.04
2019.01.05
2019.01.06
...
2019.01.30
From Individual to Relation

Features

Demographics
Behaviors
Preferences
Economic Status
Education Background

Cross-sectional Perspective

Longitudinal Perspective

Relation/Connection

"Tell me who your friends are and I’ll tell you who you are"
- Mexican Proverb -
Graph Analytics, one of the Data Scientist’s knives

- Bayesian Statistics
- Random Forest, XGBoost
- CNN, RNN, GAN
- Text Analysis, NLP
- PCA, factor analysis
- Clustering
- Regression, Logistic Regression
- t-Test, ANOVA
- Depends on business problem and data

Graph Analytics
Network: Everywhere with Everything, All the time

Social Network

* Grandjean, M. (2016)

Epidemiology

* http://www.netminer.com/community

MMO Role-Playing Game

* www.researchgate.net

Chemistry

* https://www.nature.com/articles/

Bank Risk

* https://cambridge-intelligence.com

1st Party Fraud

* www.infoglide.com

Gene

* www.researchgate.net

Manufacturing

* https://blog.trifinance.com
Use Cases - PageRank

- Measures the importance of a vertex in a graph by counting the number and quality of the links to that vertex

- Web Search
- Scientific impact of researchers
- Neuroscience
- Street and space usage

* Image from https://en.wikipedia.org/wiki/PageRank
Use Cases - Single Source Shortest Path

- Find a path to every vertex so that the sum of the weights of its constituent edges is minimized
  - Vehicle routing/ navigation
  - Degrees of separation in a social network
  - Mid-delay path in a telecommunications network
  - Plant and facility layout
  - VLSI (Very-Large-Scale Integration) design
Use Cases - Cyber-security by Graph model

- Using historical window events data to build historical graphs* of typical user behavior
  - Which machines does the user log in to?
  - Which machines does the user log in from?
  - How often?
  - In which order?

- Is this behavior typical?
  - Is it typical for this user?
  - Is it typical for someone in a particular department?
  - Is this typical for someone in the user’s job role?

- Graph models are sensitive to direction, order, and frequency.

Use Cases - Connected Component

- Calculate the **Jaccard Dissimilarity Scores** for each pair of materials
- If material X and Y are potential duplicates and material Y and Z are potential duplicates then X, Y, Z is a **connected component in the graph of all materials** and form a cluster

⇒ Connected component analysis resulted in 10% of materials identified as **potential duplicates** based on their bill of material attributes

Features for each material:
- part type
- material type & group
- product line & family
- revision key
- weld, material & coating specs
- quality matrix
- unit of measurement
- Weight
Agenda

1. Why Graph Analytics?
2. What is Graph Analytics?
3. Graph Analytics w/ MADlib
The Origin of Graph Theory

- Seven bridges of Konigsberg problem
- Leonhard Euler, a mathematician, proved that the problem has no solution

“The problem was to devise a walk through the city that would cross each of those bridges once and only once.”

Euler, 1753
What is Graph Theory?

- Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects.

- Vertex
- Node
- Point
- Actor

- Edge
- Link
- Arc
- Line

[ Directed Network Graph with Weight (example) ]

[ Terminology of Graph theory ]
# Graph Algorithms and Measures

## Graph-based Features

<table>
<thead>
<tr>
<th>Types</th>
<th>Question</th>
<th>Feasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>“What are the sub-graphs, component, communities?”</td>
<td>weakly-connected component</td>
</tr>
<tr>
<td>Structure</td>
<td>“What is the character of the network structure?”</td>
<td>Density, Diameter, Average path length, Modularity</td>
</tr>
<tr>
<td>Path</td>
<td>“What is the shortest path(distance) among vertices”</td>
<td>Single source shortest path, All pairs shortest path, Breadth-First Search</td>
</tr>
<tr>
<td>Centrality</td>
<td>“What is the most important vertices within a graph”</td>
<td>Degree (in/out, weight), Closeness, PageRank, Hub, Authority, Betweenness, Clustering coefficient</td>
</tr>
</tbody>
</table>
A Connected Component (or just Component) of an undirected graph is subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.

A dense graph is a graph in which the number of edges is close to the maximal number of edges. The opposite, a graph with only a few edges, is a sparse graph. The distinction between sparse and dense graphs is rather vague, and depends on the context.

- For Undirected simple graphs
  \[ D = \frac{|E|}{|V| (|V| - 1) / 2} \]

- For Directed simple graphs
  \[ D = \frac{|E|}{|V| (|V| - 1)} \]

E : the number of Edges,  V : the number of Vertices
Given a graph and a source vertex, the Single Source Shortest Path (SSSP) algorithm finds a path from the source vertex to every other vertex in the graph, such that the sum of the weights of the path is minimized.

**Table: Shortest paths from vertex ‘0’**

<table>
<thead>
<tr>
<th>ID</th>
<th>weight</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2 (= 1+1)</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

*weight*: The total weight of the shortest path from the source vertex to this particular vertex.
*parent*: The parent of this vertex in the shortest path from source.
The node **in-degree** is the number of edges pointing in to the node.

The node **out-degree** is the number of edges pointing out of the node.

<table>
<thead>
<tr>
<th>ID</th>
<th>In-degree</th>
<th>Out-degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Graph Algorithms and Measures - (4) Centrality

PageRank (1 / 2)

- PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

The size of each face is proportional to the total size of the other faces which are pointing to it.

\[
PR(A) = \frac{1 - d}{N} + d \left( \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots \right)
\]

- \(PR(A)\): PageRank of node A
- \(N\): the total number of Nodes
- \(L(B)\): the number of Links from node B
- \(d\): damping factor (probability, at any step, that a surfer will continue randomly clicking on links)

Graph Algorithms and Measures - (4) Centrality

PageRank (2 / 2)

\[
PR(A) = \frac{1 - d}{N} + d \left( \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots \right)
\]

Recursive calculation → converged

1st round

A

0.25

B

0.25

C

D

0.25

PR(A) = 0.25
PR(B) = 0.25
PR(C) = 0.25
PR(D) = 0.25

2nd round

A

0.427

B

0.108

C

0.214

D

0.037

PR(A) = \frac{(1-0.85)}{4} + 0.85 \times 0.25 / 2 + 0.25 / 1 + 0.25 / 3 = 0.427
PR(B) = \frac{(1-0.85)}{4} + 0.85 \times 0.25 / 3 = 0.108
PR(C) = \frac{(1-0.85)}{4} + 0.85 \times (0.25 / 2 + 0.25 / 3) = 0.214
PR(D) = (1-0.85) / 4 + 0.85 \times 0 = 0.037

Final round

A

B

C

D

0.048

0.127

0.069

0.038

\[\text{PR(A): PageRank of node A, N: the total number of Nodes, L(B): the number of Links from node B, d: damping factor (typically 0.85)}\]
**Graph Algorithms and Measures - (4) Centrality**

- Closeness of a node is a measure of centrality in a network, calculated as the sum of the length of the shortest paths between the node and all other nodes in the graph (i.e., based on All Pairs Shortest Path).

![Graph Diagram]

**Closeness**

\[
C(x) = \frac{1}{\sum_y d(y, x)}
\]

* N: The number of nodes, * d(y, x): The distance between y and x node

![All Pairs Shortest Path Table]

![Closeness Centrality Table]
Big Issue of Graph Algorithms - High Complexity

* image source: https://www.xkcd.com/399/
# Big Issue of Graph Algorithms - High Complexity

<table>
<thead>
<tr>
<th>Type</th>
<th>Algorithms/ Measures</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Weakly-Connected Component</td>
<td>$O(</td>
</tr>
<tr>
<td>Structure</td>
<td>Density</td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>Diameter</td>
<td>$O(</td>
</tr>
<tr>
<td>Path</td>
<td>All Pairs Shortest Path</td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>Single Source Shortest Path</td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>Breadth-First Search</td>
<td>$O(E + V)$</td>
</tr>
<tr>
<td>Centrality</td>
<td>In-Degree, Out-Degree</td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>Closeness Centrality</td>
<td>$O(</td>
</tr>
<tr>
<td></td>
<td>PageRank</td>
<td>$O(\log(\text{network size})/(1\text{-damping factor}))$</td>
</tr>
<tr>
<td></td>
<td>Betweenness Centrality</td>
<td>$O(</td>
</tr>
</tbody>
</table>

- Computationally Intensive - exponential to the number of vertices and edges

Graph Analysis at Scale, parallel processing with MADlib on Greenplum

* $|V|$: the number of Vertices in graph
* $|E|$: the number of Edges in graph
Agenda

1. Why Graph Analytics?

2. What is Graph Analytics?

3. Graph Analytics w/ MADlib
Tools for Graph Analytics

- Graph Analytics at Scale with Open Source MADlib on Greenplum

<table>
<thead>
<tr>
<th>Open Source</th>
<th>Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td>sna, igraph, ergm, network</td>
<td>NetMiner, UCINET</td>
</tr>
<tr>
<td>NetworkX, graph-tool, SNAP, pygraphviz</td>
<td>Graph DB</td>
</tr>
<tr>
<td>Gephi, graphviz</td>
<td>Interactive visualization focused</td>
</tr>
<tr>
<td>MADlib</td>
<td>Pivotal Greenplum</td>
</tr>
<tr>
<td>Data Size &amp; Processing</td>
<td>Big Data/Parallel Processing</td>
</tr>
</tbody>
</table>

Whether OSS or not...
Graph Analytics at Scale

Analytics Platform, GPDB

- Advanced Analytics *In Database*
- Designed for very large graphs (*billions* of vertices/edges)
- No need to move data and transform for external graph engine
  - One analytics database to deploy and manage
- Familiar SQL interface
- Combine context-based graph analytics with other content-based techniques

Graph Analytics with **MADlib on Greenplum**

- Designed for **very large graphs** (*billions* of vertices/edges)
- **No need to move data** and transform for external graph engine
  - One analytics database to deploy and manage
- **Scale Out**
- **MPP (Massively Parallel Processing)** Architecture
Apache MADlib: Big Data Machine Learning in SQL

- Open source, top level Apache project
- For PostgreSQL and Greenplum Database
- Powerful machine learning, graph, statistics and analytics for data scientists

- Open source: [https://github.com/apache/madlib](https://github.com/apache/madlib)
- Wiki: [https://cwiki.apache.org/confluence/display/MADLIB/](https://cwiki.apache.org/confluence/display/MADLIB/)
### Supervised Learning
- Neural Networks
- Support Vector Machines (SVM)
- Conditional Random Field (CRF)
- Regression Models
  - Clustered Variance
  - Cox-Proportional Hazards Regression
  - Elastic Net Regularization
  - Generalized Linear Models
  - Linear Regression
  - Logistic Regression
  - Marginal Effects
  - Multinomial Regression
  - Naïve Bayes
  - Ordinal Regression
  - Robust Variance
  - Decision Tree
  - Random Forest

### Unsupervised Learning
- Association Rules (Apriori)
- Clustering (k-Means)
- Principal Component Analysis (PCA)
- Topic Modelling (Latent Dirichlet Allocation)

### Nearest Neighbors
- k-Nearest Neighbors

---

### Graph
- All Pairs Shortest Path (APSP)
- Breadth-First Search
- Hyperlink-Induced Topic Search (HITS)
- Average Path Length
- Closeness Centrality
- Graph Diameter
- In-Out Degree
- PageRank and Personalized PageRank
- Single Source Shortest Path (SSSP)
- Weakly Connected Components

### Utility Functions
- Columns to Vector
- Conjugate Gradient
- Linear Solvers
  - Dense Linear Systems
  - Sparse Linear Systems
- Mini-Batching
- PMML Export
- Term Frequency for Text
- Vector to Columns

### Sampling
- Balanced/ Random/ Stratified Sampling

### Time Series Analysis
- ARIMA

---

### Data Types and Transformations
- Array and Matrix Operations
- Matrix Factorization
  - Low Rank
  - Singular Value Decomposition (SVD)
- Norms and Distance Functions
- Sparse Vectors
- Encoding Categorical Variables
- Path Functions
- Pivot
- Sessionize
- Stemming

### Statistics
- Descriptive Statistics
  - Cardinality Estimators
  - Correlation and Covariance
  - Summary
- Inferential Statistics
  - Hypothesis Tests
  - Probability Functions

### Model Selection
- Cross Validation
- Prediction Metrics
- Train-Test Split
Graph Representation in MADlib

[ Directed Graph (example) ]

```
<table>
<thead>
<tr>
<th>Vertex</th>
<th>Vertex Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Source Vertex</th>
<th>Dest Vertex</th>
<th>Edge Weight</th>
<th>Edge Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1.0</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>5.0</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3.0</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>8.0</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3.0</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2.0</td>
<td>...</td>
</tr>
</tbody>
</table>
```

Source: Pivot Greenplum
example: PageRank in MADlib

- Create vertex and edge tables to represent the graph

```sql
DROP TABLE IF EXISTS vertex;
CREATE TABLE vertex(
    id INTEGER
);

INSERT INTO vertex VALUES
(0),
(1),
(2),
(3),
(4),
(5),
(6);

DROP TABLE IF EXISTS edge;
CREATE TABLE edge(
    src INTEGER,
    dest INTEGER,
    user_id INTEGER
);

INSERT INTO edge VALUES
(0, 1, 1), (0, 2, 1), -- user id 1
(0, 4, 1), (1, 2, 1),
(1, 3, 1), (2, 3, 1),
(2, 5, 1), (2, 6, 1),
(3, 0, 1), (4, 0, 1),
(5, 6, 1), (6, 3, 1),
(0, 1, 2), (0, 2, 2), -- user id 2
(0, 4, 2), (1, 2, 2),
(1, 3, 2), (2, 3, 2),
(3, 0, 2), (4, 0, 2),
(5, 6, 2), (6, 3, 2);
```

* http://madlib.apache.org/docs/latest/group__grp__pagerank.html
example: PageRank in MADlib

- Compute the PageRank with All IDs

```
DROP TABLE IF EXISTS pagerank_out, pagerank_out_summary;
SELECT madlib.pagerank(
  'vertex', -- Vertex table
  'id', -- Vertex id column
  'edge', -- Edge table
  'src=src, dest=dest' -- Comma delimited string of edge arguments
  , 'pagerank_out' -- Output table of RageRank
  , NULL); -- Damping factor (default 0.85)
```

```
SELECT * FROM pagerank_out
ORDER BY pagerank DESC;
```

<table>
<thead>
<tr>
<th>id</th>
<th>pagerank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.28751816121211</td>
</tr>
<tr>
<td>2</td>
<td>0.21017119451415</td>
</tr>
<tr>
<td>3</td>
<td>0.146637377532288</td>
</tr>
<tr>
<td>4</td>
<td>0.102910437211324</td>
</tr>
<tr>
<td>5</td>
<td>0.102910437211324</td>
</tr>
<tr>
<td>6</td>
<td>0.0972746644343418</td>
</tr>
<tr>
<td>7</td>
<td>0.0525777229481976</td>
</tr>
</tbody>
</table>
A vertex with a high PageRank is usually considered more "important" or more "influential" or more "relevant" than a vertex with a low PageRank.

* PyGraphviz is a Python interface to the Graphviz graph layout and visualization package
example: PageRank in MADlib

- PageRank of vertices associated with each user by the grouping feature

```sql
DROP TABLE IF EXISTS pagerank_gr_out, pagerank_gr_out_summary;
SELECT madlib.pagerank(
    'vertex',  -- Vertex table
    'id',     -- Vertex id column
    'edge',   -- Edge table
    'src=src, dest=dest',  -- Comma delimited string of edge arguments
    'pagerank_gr_out',  -- Output table of PageRank
    NULL,  -- Default damping factor (0.85)
    NULL,  -- Default max iterations (100)
    0.00000001,  -- Threshold
    'user_id'  -- Grouping column name
);
```

```sql
SELECT * FROM pagerank_gr_out
ORDER BY user_id, pagerank DESC;
```

<table>
<thead>
<tr>
<th>user_id</th>
<th>id</th>
<th>pagerank</th>
<th>double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.278254883885528</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.201881146670752</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.142881123460599</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>0.114536378321472</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.10026745615438</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>0.10026745615438</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0.0619113553528898</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.318546250041731</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0.237868687733431</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.159148764893974</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0.111683344379718</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>0.111683344379718</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>0.8396428571428571</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>0.8214285714285714</td>
<td></td>
</tr>
</tbody>
</table>

PageRank of user id 1
PageRank of user id 2
**Personalized PageRank of vertices \{2, 4\}, for Recommendations**

```
DROP TABLE IF EXISTS pagerank_pers_out,
pagerank_pers_out_summary;
SELECT madlib.pagerank(
    'vertex' -- Vertex table
    , 'id' -- Vertex id column
    , 'edge' -- Edge table
    , 'src=src, dest=dest' -- Comma delimited string of edge arguments
    , 'pagerank_pers_out' -- Output table of PageRank
    , NULL -- Default damping factor (0.85)
    , NULL -- Default max iterations (100)
    , NULL -- Default Threshold (1/number of vertices*1000)
    , NULL -- No Grouping
    , '{2, 4}' -- Personalization vertices
);
```

```
SELECT *
FROM pagerank_pers_out
ORDER BY pagerank DESC;
```

```
SELECT *
FROM pagerank_pers_out_summary;
```

* Personalized PageRank = \((1-p)A + pE\), where ‘E’ is the list of vertices for personalized PageRank, ‘p’ is the damping factor
PageRank Performance on Greenplum w/ MADlib

Normal random graphs with mean degrees 50 edges per vertex (i.e., 5B edges in the largest case)

Greenplum cluster:
- 1 master
- 4 segment hosts with 6 segments per host

* Note: log-log scale
Execution times for synthetic graphs with varying number of vertices and edges

In Summary

- Capture the Relationship in Networks using Graph Analytics
  - Community, Structure, Path, Centrality
  - Combine context-based graph analytics with other content-based insights

- Graph analytics at SCALE with Open Source Software
  - Apache MADlib on Greenplum, massively parallel processing
One more thing...

GREENPLUM SUMMIT at PostgresConf 2019
by Pivotal
# Quick setup for toy project, GPDB & MADlib test at local env.

1. **Docker Image Pull**
   - `$ docker pull hdlee2u/gpdb-analytics`
   - `$ docker images`

2. **Docker Image Run(port 5432) -> Docker Container Creation**
   - `$ docker run -i -d -p 5432:5432 -p 28080:28080 --name gpdb-ds --hostname mdw hdlee2u/gpdb-analytics /usr/sbin/sshd -D`
   - `$ docker ps -a`

3. **To Start Greenplum Database and Use psql**
   - `$ docker exec -it gpdb-ds /bin/bash`
   - `su - gpadmin`
   - `gpstart -a`

4. **PGAdmin IV configuration for SQL query**
   - **Host**: localhost
   - **Port**: 5432
   - **Maintenance DB**: gpadmin
   - **Username**: gpadmin
   - **Password**: pivotal
   - **Group**: Servers
   - **Terminal port**: 22
   - **PGAdmin 4 download**: https://www.pgadmin.org/download/

5. **Docker Container Stop, Restart**
   - `$ docker stop gpdb-ds`
   - `$ docker start gpdb-ds`
   - `$ docker ps`

---

* [GPDB on CentOS with MADlib, PL/R and PL/Python](https://cloud.docker.com/repository/docker/hdlee2u/gpdb-analytics)
GPDB Connect by psycopg2

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

Created on Tuesday Jan 29 2019
@author: Hongdon Lee

# % network visualization using Graphviz
import pandas as pd
import pygraphviz as pgv

def run_query(query):
    import pandas as pd
    import psycopg2 as pg

    # DB Connection
    conn = pg.connect(host='localhost',
                      port='5432',
                      dbname='gpadmin',
                      user='gpadmin',
                      password='pivotal')

    # Get a DataFrame
    query_result = pd.read_sql(query, conn)
    conn.close()
    return query_result

# % Network Edge Table with PageRank score
query = "
select a.*, b.pagerank
  from edge a
    left outer join pagerank_out b on a.src = b.id;
"

edge_pagerank = run_query(query)

# % PageRank Values
query = "
select * from pagerank_out;
"

pagerank_out = run_query(query)
```

Query from GPDB using Python

* Graphviz, PyGraphviz install : https://rfriend.tistory.com/382
# NW Visualization using Graphviz with different size by PageRank
# Generating the output flow_graph with PyGraphviz
flow_graph = pgv.AGraph(strict=False, directed=True)  # directed graph

# Flow Direction(Left to Right, or Top to Bottom)
flow_graph.graph_attr['rankdir'] = 'LR'  # from Left to Right

# Node Shape
flow_graph.node_attr['shape'] = 'circle'

# Making node with different size proportional to PageRank
for i in range(len(pagerank_out)):
    label_text = str(pagerank_out.id[i]) + '
(' + str(pagerank_out.pagerank[i].round(decimals=2)) + ')
    node_width = pagerank_out.pagerank[i]*10
    node_height = pagerank_out.pagerank[i]*10

    flow_graph.add_node(str(pagerank_out.id[i]),
                        label=label_text,
                        **{'width': str(node_width),
                           'height': str(node_height)})

# Adding edge with different color by user_id
colors = ['blue', 'red']

for i in range(len(edge_pagerank)):
    if edge_pagerank.user_id[i] == 1:
        color_text = colors[0]
    else:
        color_text = colors[1]

    flow_graph.add_edge(str(edge_pagerank.src[i]),
                        str(edge_pagerank.dest[i]), color=color_text)

# Finally, Draw the Network Diagram using dot program :-)  
flow_graph.draw("/Users/ihongdon/Documents/nw_diagram.png", prog='dot')
Pivotal

Transforming How The World Builds Software