E6893 Big Data Analytics Lecture 8:

*Linked Big Data — Graph Computing (II)*

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

IBM Chief Scientist, Graph Computing

October 27th, 2016
Final Project Proposal Presentation

Date/Time: November 17, 7:00pm - 10:00pm

Each Team — about 3 mins:

1. Team members and expected contributions of each member;
2. Motivation of your project (The problem you would like to solve);
3. Dataset, algorithm, and tools for your project;
4. Current Status of your project.

Please update your team info in the Project webpage by November 10. The presentation schedule will be announced on November 17.

The website will be opened to allow you upload your slides by November 16.

If a project is purely by CVN students, please submit your slides without oral presentation.
Graph Definitions and Concepts

- A graph:
  \[ G = (V, E) \]

- \( V = \) Vertices or Nodes
- \( E = \) Edges or Links

- The number of vertices: “Order”
  \[ N_v = |V| \]
  \[ N_e = |E| \]
In-degrees and out-degrees

- For Directed graphs:

```
In-degree = 8
```
```
Out-degree = 8
```
Degree Distribution Example: Power-Law Network


$p_k = e^{-m} \cdot \frac{m^k}{k!}$

$p_k = C \cdot k^{-\tau} e^{-k/k}$

Newman, Strogatz and Watts, 2001
Another example of complex network: Small-World Network

Six Degree Separation:
adding long range link, a regular graph can be transformed into a small-world network, in which the average number of degrees between two nodes become small.

from Watts and Strogatz, 1998

C: Clustering Coefficient, L: path length, 
(C(0), L(0)): (C, L) as in a regular graph 
(C(p), L(p)): (C,L) in a Small-world graph with randomness $p$. 
Some examples of Degree Distribution

(a) scientist collaboration: biologists (circle) physicists (square), (b) collaboration of movie actors, (d) network of directors of Fortune 1000 companies
Network size is positively correlated with performance.
- Each person in your email address book at work is associated with $948 dollars in annual revenue.

$74.07 increase in monthly revenues or $948 annual revenues

Std error = (26.38)***
Significant at p < 0.01
Centrality

“There is certainly no unanimity on exactly what centrality is or its conceptual foundations, and there is little agreement on the procedure of its measurement.” – Freeman 1979.

Degree (centrality)
Closeness (centrality)
Betweenness (centrality)
Eigenvector (centrality)
Closeness: A vertex is ‘close’ to the other vertices

\[ C_{CI}(v) = \frac{1}{\sum_{u \in V} \text{dist}(v, u)} \]

where dist(v,u) is the geodesic distance between vertices v and u.
Betweenness $\Rightarrow$ Bridges

Example: Healthcare experts in the world

Connections between different divisions

Example: Healthcare experts in the U.S.

Key social bridges
Betweenness measures are aimed at summarizing the extent to which a vertex is located ‘between’ other pairs of vertices.

Freeman’s definition:

\[ C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma(s, t \mid v)}{\sigma(s, t)} \]

Calculation of all betweenness centralities requires calculating the lengths of shortest paths among all pairs of vertices.
Computing the summation in the above definition for each vertex.
Eigenvector Centrality

Try to capture the ‘status’, ‘prestige’, or ‘rank’.
More central the neighbors of a vertex are, the more central the vertex itself is.

\[ c_{Ei}(v) = \alpha \sum_{\{u,v\} \in E} c_{Ei}(u) \]

The vector \( c_{Ei} = (c_{Ei}(1), \ldots, c_{Ei}(N_v))^T \) is the solution of the
eigenvalue problem:

\[ A \cdot c_{Ei} = \alpha^{-1} c_{Ei} \]
PageRank Algorithm (Simplified)
IBM System G Graph Computing Tools

Visualization
- Huge Network Visualization
- Network Propagation
- Dynamic Network
- Geo Network Visualization
- Graphical Model

Analytics
- Graph Computing Tools APIs and Query Language Support
  - Communities
  - Centralities
  - Ego Features
  - Graph Search
  - Graph Matching
  - Shortest Paths
  - Bayesian
  - Deep Learning
  - SpatioTemporal
  - Anomaly Detection

Middleware
- In-Memory Graph RT Library
- Multi-Core Multi-Thread Graph RT Library
- Distributed Memory Graph RT Library
- GPU Graph Computing Driver

Database
- Graph Data Interface (TinkerPop)
- Spark Interface
- Streams Interface
- Enterprise Graph Database Enhancer
- Other Graph Store
  - GBase
  - HBase
- Performance Driven System G Native Graph Store

Hardware
- File System (Linux FS, Hadoop HDFS, etc.)
- Server (Linux & OS X)
- Cluster (CPU, CPU+GPU)
- Cloud
- Mobile (iOS)
- Mainframe (System Z & Power)
- Super Computer

© 2016 CY Lin, Columbia University
System G – Graph Computing as an Intelligence Machine Machine

Graph Database
- Property Graph

Graph Analytics
- Relation Graph

Graphical Models
- Reasoning Graph

Memory

Perception

Intelligence

Related Information
- IBM System G – Graph Computing as an Intelligence Machine Machine
  - http://systemg.research.ibm.com

Contextual Analysis

Machine Reasoning & Deep Learning

© 2016 CY Lin, Columbia University
Machine Reasoning

Perception → Abstract comprehension → Cognitive

Sensor Layer | Feature Observation Layer | Concept Perception Layer | Semantic Abstract Layer | Cognitive Reasoning Layer
---|---|---|---|---
Observation | Memory | Perception | Abstract comprehension | Cognitive

Cogni...
IBM System G Application Use Cases

1. System G for Expertise Location
2. System G for Recommendation
3. System G for Commerce
4. System G for Financial Analysis
5. System G for Social Media Monitoring
6. System G for Telco Customer Analysis
7. System G for Watson
8. System G for Data Exploration and Visualization
9. System G for Personalized Search
10. System G for Anomaly Detection (Espionage, Sabotage, etc.)
11. System G for Fraud Detection
12. System G for Cybersecurity
13. System G for Sensor Monitoring (Smarter another Planet)
14. System G for Cellular Network Monitoring
15. System G for Cloud Monitoring
17. System G for Traffic Navigation
18. System G for Image and Video Semantic Understanding
19. System G for Genomic Medicine
20. System G for Brain Network Analysis
21. System G for Data Curation
22. System G for Near Earth Object Analysis
System G Graph Analytical Tools

- **Network topological analysis** tools
  - Centralities (degree, closeness, betweenness)
  - PageRank
  - Communities (connected component, K-core, triangle count, clustering coefficient)
  - Neighborhood (egonet, K-neighborhood)

- **Graph matching and search** tools
  - Graph search/filter by label, vertex/edge properties (including geo locations)
  - Graph matching
  - Collaborative filtering

- **Graph path and flow** tools
  - Shortest paths
  - Top K-shortest paths

- **Probabilistic graphical model** tools
  - Bayesian network inference
  - Deep learning
Characteristics of IBM System G Graph Analytics

• Cover a wide range of graph analytics to support many application use cases in different domains, e.g.:
  • Enterprise social network analysis, expertise search, knowledge recommendation
  • Financial/security anomaly/fraud detection
  • Social media monitoring and analysis
  • Cellular network analytics in Telco operation
  • Patient and disease analytics for healthcare
  • Live neural brain network analysis
• Provide efficient in-memory computation as well as on-disk persistence
• Optimal performance enabled by IBM System G graph database technologies that focus on efficient use of available computing resources with architecture-aware design to leverage system/architecture advantages
• Single-threaded, concurrent (shared memory), and distributed versions
• Multiple deployment options to suit different customer preferences and needs
  • C++ executables in Linux environments (Redhat CentOS, Ubuntu, Mac OS X, Power)
  • TinkerPop (Blueprints) API
  • gShell (a shell-like environment with interactive, batch, and server/client modes to operate multiple graph stores simultaneously)
  • Gremlin console
  • REST API Web service
  • Python wrapper
Download IBM System G Standard Edition (on-premise)

http://systemg.research.ibm.com/download.html

or

Existing foundation of 16 types of graph visualization assets in these 4 categories:

- Multivariate Graphs: nodes and edges have multivariate attributes. E.g., healthcare graphs, workflow graphs, behavior reasoning graphs, etc.
- Heterogeneous Graphs: graphs in which nodes and edges are in different categories and types. E.g.: bipartite/tripartite/multi-partite graphs, geospatial graphs, etc.
- Dynamic Graphs: graphs whose topology and attributes change over time. E.g., relationship graphs, information propagation graphs, etc.
- Big Graphs: graphs with millions or even billions of nodes and edges. Hierarchical-based visualization or infinite-plane based visualization. E.g., social graphs, knowledge graphs, etc.

Web-based:
HTML5
WebGL

http://systemg.ibm.com
Resources

- IBM System G on Bluemix (need registration)
  - http://systemg.mybluemix.net

- IBM System G Graph Analytics Overview

- IBM System G Graph Tools Trial Download

- IBM System G Graph Tools Installation Guide and Documentation
IBM System G Graph Tools - Overview

Overview

IBM System G Graph Tools provide a set of tools for developers and end users to create graph stores, conduct graph queries, run graph analytics, and explore graphs via interactive visualizations. They are built on top of IBM System G Native Graph Store and Middleware specifically developed for high-performance graph computing based on a property graph model.

IBM System G Graph Tools Trial Download (1.2.2) provides the following tools in a single-node environment:

- gShell (stand-alone): a shell-like environment with a set of commands for creating, updating, querying multiple graph stores and running graph analytics
- REST API service (dependent on gShell): an enhanced version of TinkerPop 2 Rexster Basic REST API to operate graph stores via gShell commands
- Python interface to gShell: Python wrappers of all gShell commands
- Blueprints (2.5.0) API (stand-alone): for operating graph stores in Java applications
- Gremlin (2.4.0) console (stand-alone): for creating and traversing graphs using the Gremlin-Groovy graph query language
- IBM System G Lite (dependent on REST API service): a Web application for creating, querying, and exploring graphs through GUI and interactive visualizations

Packages for CentOS (Redhat) 6.5, Ubuntu 14.04, Power 8 and Mac OS X are currently available.

Note: A new version (1.3.0) will be available on Friday 11/6 or Monday 11/9. HW#3 will be delayed to 11/19.

2. In the middle of webpage, find download and support section

---

**Download and Support**

The System G Graph Tools Trial Download version is free, intended for experimentation, research and application development. You can use it to support your commercial or non-commercial applications. But, please note that, this software cannot be redistributed or sold. It is the users' own risk using the software.

You can download the IBM System G Graph Tools Trial Download from [here](http://systemg.research.ibm.com/download.html). Click it download

There is no online support for this version and IBM may choose to update the version at our discretion. Feedback & enhancement suggestions may be sent to systemg @ us . ibm . com (remove white space).
IBM System G Graph Tools - Download

This is temporary page, you can download directly.
IBM System G Graph Tools - Download

Download from IBM developerWorks:
1. You need a IBM id to download the package
2. If you have, just log in, or register one
IBM id registration page:
1. Fill out the form to get your IBM id
Accept usage terms to download IBM System G Graph Tools
This is old page, now, we support Linux (CentOS and Ubuntu), Mac OS (Yosemite), and IBM Power.

This is old version, new release is 1.3.0 (will be available on developer work soon)
IBM System G Graph Tools - Installation

All following instructions are used for Ubuntu 14.04, installation of some packages might be varied based on your OS.

Installation

After downloading the package, simply untar the .tar.gz file (tar -xvzf), then follow a few steps here to set up the tools in your environment.

IBM System G Graph Tools Setup

After downloading the .tar.gz package file, use tar -xvzf to untar it. The unpacked directory contains the following files/sub-directories:

- README: a text file that describes the content of the package and provides references to documentation files
- systemg.sh: a script to set up environment variables required to run IBM System G Graph Tools
- doc/: documentation files
- data/: sample data files for tests
- gshell/: gShell executable files, sample data, and test scripts
- lib/: library files for gShell
- python/: Python interface to gShell
- blueprints-gremlin/: Blueprints API and Gremlin
- resapi/: REST API executable files and scripts
- systemg-lite/: IBM System G Lite visualization

IBM System G Graph Tools - Installation

Package step

Set path

Most components of IBM System G require LD_LIBRARY_PATH to locate the library files needed for execution. To get started, first open systemg.sh to edit the path of `<systemg_home_path>` (the absolute path to the unpacked directory) according to the setup in your environment, then copy the file to `/etc/profile.d/`. You need to log out and back in again for the script to be executed to set the path.

If there isn’t `/etc/profile.d/` in your environment, do the following before copying systemg.sh:

1. Create `/etc/profile.d/`, e.g. `sudo mkdir /etc/profile.d`
2. Edit `/etc/profile` to add the following to the bottom:

   ```bash
   for sh in /etc/profile.d/*sh ; do
       [ -r "$sh" ] && . "$sh"
   done
   unset sh
   ```

3. Log out and back in.

   If you don’t have permission to put systemg.sh under `/etc/profile.d/`, the command listed in systemg.sh needs to be executed manually to set LD_LIBRARY_PATH properly.

You are ready to go for gShell with command line and Python interface.
gShell

gShell is a shell-like environment on top of IBM System G Graph Store. It allows users to create, update and query multiple graph stores simultaneously and supports both directed and undirected graphs.

Interactive mode

./gShell interactive

gShell displays a prompt >> waiting for input commands. To exit, type close_all or Ctrl-c.

Batch mode

./gShell interactive < <batch_file> (e.g., ./gShell interactive < script.txt)

The batch mode is a variant of the interactive mode, where the commands stored in a text file <batch_file> are redirected to the interactive mode (using <) and executed sequentially. Note that close_all needs to be included as the last command in the file in order to exit from gShell.

All gShell commands share the same format:

command --arg1 value1 --arg2 value2 ...

IBM System G Graph Tools - gShell

`./gShell` interactive

double click “tab” to show possible auto-completion
IBM System G Graph Tools - gShell

Create a simple graph:
- ./gShell interactive
- delete --graph g
- create --graph g --type directed

- add_vertex --graph g --id Alice --label user
- add_vertex --graph g --id 1234 --label order --propwithtype date:string:20120808 status:string:delivered
- add_vertex --graph g --id 5678 --label order --propwithtype date:string:20120816 status:string:delivered
- add_vertex --graph g --id abcd --label item --prop description:"strawberry ice cream" handling:freezer
- add_vertex --graph g --id efab --label item --prop description:"brussels sprouts"
- add_vertex --graph g --id cdef --label item --prop description:"expresso beans"

- add_edge --graph g --src Alice --targ 1234 --edgelabel _ --prop type:PLACED time:MOST_RECENT
- add_edge --graph g --src Alice --targ 5678 --edgelabel _ --prop type:PLACED
- add_edge --graph g --src 5678 --targ cdef --edgelabel _ --prop type:CONTAINS
- add_edge --graph g --src 1234 --targ abcd --edgelabel _ --prop type:CONTAINS
- add_edge --graph g --src 1234 --targ efab --edgelabel _ --prop type:CONTAINS
- add_edge --graph g --src 1234 --targ cdef --edgelabel _ --prop type:CONTAINS
- close_all
gShell with Python interface

1. Download and install the respective version of Python if it is not already available in your environment.
2. Copy the files from the sub-directory corresponding to the chosen Python version (i.e. python2.6, python2.7, python3.4) to the parent directory python/.
3. Make sure LD_LIBRARY_PATH is set properly by following the instruction in the "Set path" section above.
4. Test if the setup is successful by typing:

   ```
   $ cd python
   $ python -c "import py_gShell"
   ```

   If nothing is returned, the setup is successful. Otherwise, make sure that both _py_gShell.so and py_gShell.py are copied to the python/ directory.
5. Run the test scripts provided.
6. Put "from py_gShell import _py_gShell as gShell" in your Python code, and you are good to go!

IBM System G Graph Tools - gShell with Python interface

```python
#!/usr/bin/python
from py_gShell import _py_gshell as gShell
import json

g = gShell()
g.delete_graph("testu")
g.create_graph("testu", "undirected")
g.load_csv_vertices(csvfile="data/test.vertices.dat", keypos=0, labelpos=1)
g.load_csv_edges(csvfile="data/test.edges.dat", srcpos=0, targpos=1, labelpos=3)
g.add_vertex(vertex_id="7", label="C", prop={"tag":"T2","value":0.1})
g.add_vertex(vertex_id="8", prop={"value":0.4})
g.add_vertex(vertex_id="9", label="C", prop={"value":0.5})
g.update_vertex(vertex_id="9",prop={"value":0.55,"other":"1"})
g.add_edge(src="7", targ="8", edgelabel="c")
g.add_edge(src="7", targ="1", edgelabel="c",prop={"weight":8.0})
g.update_edge(src="1",targ="7", prop={"weight":8.6, "other":"2"})
g.add_edge(src="8",targ="9")
g.update_edge(src="1",targ="2", prop={"weight":6.5})
g.analytic_start_engine(edgeweightpropname="weight")
print json.dumps(json.loads(g.analytic_find_path(src="1",sink="2")), indent = 4)

print json.dumps(json.loads(g.analytic_find_path(src="1",sink="2",label="b")), indent = 4)
g.analytic_stop_engine()
```

Output of the above Python script:

```
{  
  "paths": [ 
    { 
      "src": "1",  
      "path": "1\n1\n2",  
      "sink": "2",  
      "distance": 1.0  
    },  
    "time": [ 
      "TIME": "3.31402e-05"  
    ]  
  ]  
}
```

```
{  
  "paths": [ 
    { 
      "src": "1", 
      "path": "1\n3\n5\n2", 
      "sink": "2", 
      "distance": 3.0  
    },  
    "time": [ 
      "TIME": "2.00808e-05"  
    ]  
  ]  
}
```
Create a simple graph:

```python
from py_gShell import _py_gshell as gShell
import json
g = gShell()
g.delete_graph("g")
g.create_graph("g", graph_type="directed")

g.add_vertex(vertex_id="Alice", label="user")
g.add_vertex(vertex_id="1234", label="order", prop={"date":"20120808", "status":"delivered"})
g.add_vertex(vertex_id="5678", label="order", prop={"date":"20120806", "status":"delivered"})
g.add_vertex(vertex_id="abcd", label="item", prop={"description":"strawberry ice cream", "handling":"freezer"})
g.add_vertex(vertex_id="efab", label="item", prop={"description":"brussels sprouts"})
g.add_vertex(vertex_id="cdef", label="item", prop={"description":"expresso beans"})

g.add_edge(src="Alice", targ="1234", edgelabel="_", prop={"Type":"PLACED"})
g.add_edge(src="Alice", targ="5678", edgelabel="_", prop={"type":"PLACED", "time":"MOST_RECENT"})
g.add_edge(src="5678", targ="1234", edgelabel="_", prop={"time":"PREVIOUS"})
g.add_edge(src="1234", targ="abcd", edgelabel="_", prop={"type":"CONTAINS"})
g.add_edge(src="1234", targ="efab", edgelabel="_", prop={"type":"CONTAINS"})
g.add_edge(src="1234", targ="cdef", edgelabel="_", prop={"type":"CONTAINS"})
g.add_edge(src="5678", targ="cdef", edgelabel="_", prop={"type":"CONTAINS"})
```
IBM System G Graph Tools - gShell with Python interface

Graph analytics:
For graph analytics in System G Graph Tools, it will return with the following fields, and all results will be written back to vertices/edges as a property. (it is easy to retrieve in the future.)

1. time: execution time
2. key field: major results (field name depends on analytics)
3. summary: if there are some statistic results, they will be stored in this field.
4. others: supplementary information

```
print json.dumps(g.analytic_shortest_paths(src=user98, sink=user34), indent=2)

[
  "paths": [
    {"path": "user98--user161--user125--user34 | user98--user183--user113--user34 | user98--user72--user77--user34"},
    {"num_paths": 3, "sink": "user34", "distance": 3.0}
  ],
  "time": {
    "TIME": "0.090428889"
  }
]
```
Another example: count number of triangular in the graph

```json
json_print(g.analytic_triangle_count())
```
Interact with Python: (through JSON library) since every result will be returned as “JSON string”, we can convert it as JSON object to interact with user-defined functions.

Example: with graph analytic tool to obtain connected components, and then get (I) number of connected components, (II) size of the maximal connected component, and (III) one of the vertex in the connected component.

```
ret = json.loads(g.analytic.connected_component())
print(len(ret['summary']))
max_component = 0
max_component_label = ''
for summary in ret['summary']:
    if int(summary['count']) > max_component:
        max_component = int(summary['count'])
        max_component_label = summary['label']
print(max_component)
print(max_component_label)
```

Results:

49
144
user1
Demo

gShell with Python Interface
Open Source TinkerPop Stack (Apache Incubator)

http://tinkerpop.incubator.apache.org

http://sql2gremlin.com
Graph Language

Gremlin is a graph traversal language. The documentation herein will provide all the information necessary to understand how to use Gremlin for graph query, analysis, and manipulation. Gremlin works over those graph databases/frameworks that implement the Blueprints property graph data model. Gremlin is a style of graph traversal that can be used in various JVM languages. This distribution of Gremlin provides support for Java and Groovy. Except where otherwise stated, the documentation herein is respective of the Groovy implementation (minor syntactic tweaks are required to map the ideas over to other JVM implementations).

Please join the Gremlin users group at http://groups.google.com/group/gremlin-users for all TinkerPop related discussions.¹

```java
// calculate basic collaborative filtering for vertex 1
m = [:]
g.v(1).out('likes').in('likes').out('likes').groupCount(m)
m.sort{-it.value}

// calculate the primary eigenvector (eigenvector centrality) of a graph
m = [:]; c = 0;
g.V.as('x').out.groupCount(m).loop('x',c++ < 1000)
m.sort{-it.value}
```
IBM System G Graph Tools - Gremlin-gShell

Gremlin

Gremlin is a domain-specific language for traversing property graphs. JDK 1.7+ is required. The package supports Gremlin 2.4.0 in two modes: command-line and browser-based.

A few external library files need to be downloaded by running the following script before starting Gremlin:

```
$ cd blueprints-gremlin
$ ./bin/download_dependencies.sh
```

The command-line Gremlin console can be started by simply running `gremlin.sh` in the `blueprints-gremlin/bin/` directory.

```
$ cd blueprints-gremlin
$ ./bin/gremlin.sh

```

To exit the Gremlin console, simply enter `quit`.

Install Java and unbuffer: sudo apt-get install openjdk-7-jdk expect-dev

Blueprint API: [http://www.tinkerpop.com/docs/javadocs/blueprints/2.5.0/com/tinkerpop/blueprints/Graph.html](http://www.tinkerpop.com/docs/javadocs/blueprints/2.5.0/com/tinkerpop/blueprints/Graph.html)
**IBM System G Graph Tools - Gremlin-gShell**

```groovy
gremlin> g = CreateGraph.openGraph("nativemem_authors","awesome")
==>nsgraph[vertices:7 edges:8]
gremlin> g.class
class com.ibm.research.systemg.nativestore.tinkerpop.NSGraph
gremlin> // lets look at all the vertices
==>true
gremlin> qs = new GShell()
gremlin> qs.execute("create --graph test --type directed")
140711320353584
[create] [--graph] [test] [--type] [directed]
=>{
  "info":{"MESSAGE":"store [test] is created!"}}
gremlin> qs.execute("add_vertex --graph test --id \"test node\" --prop tag:\"test tag\"")
139868232521952
[add_vertex] [--graph] [test] [--id] [test node] [--prop] [tag] [test tag]
=>{
  "info":{"MESSAGE":"vertex is added"},
  "time":{"TIME":"0.000422001"}}
```
IBM System G Graph Tools - Installation

REST API

To enable REST API service, copy gshell script, gShellClientMulti, invoke_with_load_library.sh, and lib/ in a <directory> under the cgi-bin directory of the Apache HTTP Server (e.g. /var/www/cgi-bin/restapi/). The URLs of all API calls will have a prefix of http://<host>/cgi-bin/<directory>/gshell. If the port number used by sysgSuperMgr is not 6688, and/or sysgSuperMgr is located on a different machine, the IP address and port number specified in line #169 of gshell script (search for "gShellClientMulti") need to be modified accordingly. Use chmod 755 * to set proper file permissions.

You need to install apache http server and perl parser

After the above set-up, run sysgSuperMgr to start REST API service:

$ cd gshell
$ ./sysgSuperMgr <port_number> <max_num_active_users>

For example:

$ nohup ./sysgSuperMgr 6638 10 > sysg.out 2> sysg.err < /dev/null &

To stop REST API service safely, go to the directory where gShellClientMulti is located, run:

$ ./gShellClientMulti "bye" admin <host_ip> <port_number>

For example:

./gShellClientMulti "bye" admin 127.0.0.1 5688

IBM System G Graph Tools - Installation

REST API - Apache server installation

1. Install apache server: sudo apt-get install apache2
2. Activate your cgi-bin: sudo a2enmod cgi
3. Add your ServerName: sudo emacs /etc/apache2/apache2.conf
   1. Add ServerName at the end of line, “ServerName 127.0.0.1”
4. Change your cgi-bin path to /var/www/cgi-bin: (in Ubuntu, default path is /usr/lib/cgi-bin (if /var/www/cgi-bin did not exist, mkdir before you change cgi-bin path)
   1. sudo emacs /etc/apache2/conf-available/serve-cgi-bin.conf
      • Find ScriptAlias: change to "/cgi-bin/ /var/www/cgi-bin/"
      • Find <Directory>, change to "/var/www/cgi-bin"
5. Change your Document root: sudo emacs /etc/apache2/sites-available/000-default.conf
   • Find DocumentRoot, change to “/var/www”
6. Restart Apache2 server: sudo service apache2 restart
IBM System G Graph Tools - Installation

REST API - Perl parser installation

1. Install perl: sudo apt-get install perl
2. install other perl modules via CPAN Shell: sudo perl -MCPAN -e shell (type yes twice)
   1. install Bundle::LWP
   2. install JSON
   3. install CGI
System G Lite

IBM System G Lite requires an HTTP server with PHP version 5.0+ and the REST API service described earlier.

To get started, perform the following steps:

1. Copy `systemg-lite/` to the Web hosting directory (e.g. `/var/www/`) so that IBM System G Lite can be accessed through `http://<host>/systemg-lite`
2. Open `systemg-lite/php/config.php` and change the URL in the line of define("GRAPHDB_URL", ... to be the URL of the REST API service described earlier
3. Use `chmod 755` for all directories and files under `systemg-lite/`
4. Use `chmod 777` for `systemg-lite/trp/` and `systemg-lite/upload/files/` directories (make sure the directories exist)

You need to install `php5`: `sudo apt-get install php5`

IBM System Visualizer (SystemG-Lite)
IBM System G Visualizer – Graph Data Explorer

Visual Query Panel

Visualization Panel

Visual Mapping Panel

Console Panel
Panel Introduction

- Visual Query Panel
  - Providing users a friendly UI to create, delete, and query graphs from the System G native store.

- Console Panel
  - Display all the interaction information with System G native store.
  - Execute user defined query.

- Visualization Panel
  - Rendering graph structure on screen for users to visually explore graphs.

- Visual Mapping Panel
  - Customizing rendering effects to show desired graph information.
Visual Query Panel – Creating a graph

1: Click “Create Graph”; 2: Prepare the graph data; 3: Set the graph name; 4: Upload node files; 5: Upload edge files and finalize creating the graph.
“analytics_degree <= 10 and (group == “center” or group == “guard”)"
Console Panel – User typed query

find_vertex_max_degree --graph Basketball --edgetype all

Query with no graph returned

get_eigonet --graph imdb_with_degree --id "Tom Hanks" --depth 2

Query with graph returned
Visual Mapping Panel

<table>
<thead>
<tr>
<th>Name</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Color</td>
<td>Change the background color of the canvas.</td>
</tr>
<tr>
<td>Node Default Color</td>
<td>Set a unified color for all nodes.</td>
</tr>
<tr>
<td>Edge Default Color</td>
<td>Set a unified color for all edges.</td>
</tr>
<tr>
<td>Show Nodes</td>
<td>Set the visibility of all nodes.</td>
</tr>
<tr>
<td>Node Color Mapping</td>
<td>Assign color to nodes according to selected property of nodes.</td>
</tr>
<tr>
<td>Node Size Mapping</td>
<td>Assign the radius of nodes according to selected property of nodes.</td>
</tr>
<tr>
<td>Filter Node Label by Node Size</td>
<td>Selectively show the node label according to the threshold. Labels will be shown for the nodes of which the size is larger than the threshold.</td>
</tr>
<tr>
<td>Node Label Mapping</td>
<td>Set the label value according to selected property of nodes.</td>
</tr>
<tr>
<td>Node Label Size</td>
<td>Adjust the font size of node labels.</td>
</tr>
<tr>
<td>Show Edges</td>
<td>Set the visibility of all edges.</td>
</tr>
<tr>
<td>Edge Color Mapping</td>
<td>Assign color to edges according to selected property of edges.</td>
</tr>
<tr>
<td>Edge Label Mapping</td>
<td>Set the label value according to selected property of edges.</td>
</tr>
<tr>
<td>Edge Label Size</td>
<td>Adjust the font size of edge labels.</td>
</tr>
<tr>
<td>Edge Thickness Mapping</td>
<td>Assign thickness to edges according to selected property of edges.</td>
</tr>
<tr>
<td>Edge Style</td>
<td>Select the rendering style of edges. For directed graphs, users also can choose if showing the arrows or not.</td>
</tr>
</tbody>
</table>
Visualization Panel – After Customization

Node Color Mapping: Actor, Movie, undefined, Male Actor
Edge Color Mapping: ACTS_IN

[Image of a complex network visualization with nodes and edges indicating relationships between actors and movies.]

© 2016 CY Lin, Columbia University
Users can further specify colors by clicking the color blocks shown in the legend area.
http://systemg.ibm.com/tool/visualizer/
IBM System G Eco-System (GraphBIG)
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

**Fetch Code**

**Code:** https://github.com/graphbig/graphBIG

**Doc:** https://github.com/graphbig/GraphBIG-Doc

```
-bash:~$ git clone https://github.com/graphbig/graphBIG.git
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 10 bytes/s, done.
Resolving deltas: 100% (229/229), done.
Checking connectivity... done.
-bash:~$ 
```

**Breakdown of Execution Cycles**

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

<table>
<thead>
<tr>
<th>Bfs</th>
<th>Dfs</th>
<th>Kcore</th>
<th>Comp</th>
<th>SPath</th>
<th>DCentr</th>
<th>Bcentr</th>
<th>TC</th>
<th>Gibbs</th>
<th>GCons</th>
<th>Gup</th>
<th>TMorph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

- **Backend**
- **Retiring**
- **BadSpeculation**
- **Frontend**
76% of the total execution is spent inside the framework by invoking primitive graph operations framework. The framework actually plays a critical role.
CSR format is compact, and maybe good for cache performance. But it is static, and cannot support structure changes. However, in practices, graphs are usually dynamic. This is why vertex-centric representation is popular across multiple graph frameworks.
Graph Computing Types

• Computation on graph structure (CompStruct)
  • Example: Breadth-first search
  • Irregular access pattern, heavy read access

• Computation on dynamic graph (CompDyn)
  • Example: Streaming Graph
  • Dynamic graph structure, dynamic memory usage

• Computation on graph property (CompProp)
  • Example: Belief propagation
  • Heavy numeric operations on graph property
We start from the use cases of IBM System G. By analyzing the use cases, we are able to summarize the computation and data types. Meanwhile, we select workloads and data from them. After that, we then have a reselection stage. In the reselection stage, we reselect workloads and data to ensure that they cover all computation and data types.
In total, we analyzed 21 use cases from 6 different categories, from science exploration to security. Different categories contain different use cases and different selected workloads also have different popularities across the use cases. But in general, all workloads are widely used in multiple real-world use cases.
## Workload Summary and Experiments to Show

<table>
<thead>
<tr>
<th>Category</th>
<th>Workload</th>
<th>Computation Type</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph traversal</strong></td>
<td>BFS</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>CompStruct</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Graph update</strong></td>
<td>Graph construction (GCons)</td>
<td>CompDyn</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graph update (GUp)</td>
<td>CompDyn</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Topology morphing (TMorph)</td>
<td>CompDyn</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Graph analytics</strong></td>
<td>Shortest path (5Path)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>kCore</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Connected component (CComp)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Graph coloring (GColor)</td>
<td>CompStruct</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Triangle counting (TC)</td>
<td>CompProp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gibbs Inference (GI)</td>
<td>CompProp</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Social analytics</strong></td>
<td>Betweenness Centrality (BCentr)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Degree Centrality (DCentr)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>Type</th>
<th>Vertex #</th>
<th>Edge #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter Graph</td>
<td>Type 1</td>
<td>120M</td>
<td>1.9B</td>
</tr>
<tr>
<td>IBM Knowledge Repo</td>
<td>Type 2</td>
<td>154K</td>
<td>1.72M</td>
</tr>
<tr>
<td>IBM Watson Gene Graph</td>
<td>Type 3</td>
<td>2M</td>
<td>12.2M</td>
</tr>
<tr>
<td>CA Road Network</td>
<td>Type 4</td>
<td>1.9M</td>
<td>2.8M</td>
</tr>
<tr>
<td>LDBC Graph</td>
<td>Synthetic</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>
GraphBIG Hands-on

Fetch Code

Code:  https://github.com/graphbig/graphBIG
Doc:  https://github.com/graphbig/GraphBIG-Doc

GraphBIG is open sourced under BSD license. We have an organization in github named as graphbig. To obtain the GraphBIG code is pretty simple. Just do use git to perform a "git clone"
More detailed documents can also be found in a separate repository in the same organization in github.
GraphBIG Hands-on - 2

Compile

Require: gcc/g++ (>4.3), gnu make
Just “make all”

GraphBIG is a standalone package. It doesn’t require any external libraries. But of course, you need a gcc and for gpu workloads, you need cuda sdk
To compile it, just “make all”.
To compile the full suite, you can “make all” at the top level. If you just want to compile CPU benchmarks, get into “benchmark/” directory and “make all”. Similarly for GPU workloads, get into “gpu_bench/” and “make all”
Test Run

Just **`make run`**
Using default “small” dataset

Help info:  `./<exe> --help`

It is also pretty simple to make a test run of GraphBIG workloads. We include the simple test run already in the makefile. You can get into the directory of any benchmark and use “make run”. Then, a test run will be performed and the output will be stored in a log file named “output.log”

To get more info about the arguments of a specific benchmark, just run it with “--help”
Characterization

Methodology
Real machine + hardware performance counters

CPU: linux perf event kernel calls (integrated with benchmarks)
GPU: CUDA nvprof

<table>
<thead>
<tr>
<th>Processor</th>
<th>Type</th>
<th>Xeon E5-2670</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>2.6 GHz</td>
<td></td>
</tr>
<tr>
<td>Core #</td>
<td>2 sockets x 8 cores x 2 threads</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>32KB L1, 256KB L2, 20MB L3</td>
<td></td>
</tr>
<tr>
<td>Memory BW</td>
<td>51.2 GB/s (DDR3)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU</th>
<th>Type</th>
<th>Nvidia Tesla K40</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Core</td>
<td>2880</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>12 GB</td>
<td></td>
</tr>
<tr>
<td>Memory BW</td>
<td>288 GB/s</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>Core-745 MHz, mem-3 GHz</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Memory</th>
<th>192 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>2 TB HDD</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>RHEL 6</td>
<td></td>
</tr>
</tbody>
</table>
We breakdown the total execution time into four categories. Both frontend and backend represent the CPU stall cycles. One is stall cycles caused by frontend issues, the other is stall cycles caused by backend issues. Badspeculation represents the wasted cycles because of wrong branch predictions. The retiring is the actual running and useful cycles.

We can see that for most workloads, backend is the dominant, it is the bottleneck here. Backend may include instruction execution, retiring, memory sub-systems. But outliers also exist, for TC (triangle counting) and Gibbs (gibbs inference), they are not suffering from backend issues. It shows an interesting diversity across benchmarks.
IBM System G Eco-System (ScaleGraph)
ScaleGraph — an Open Source version of IBM System G

ScaleGraph

ScaleGraph
A graph library for large-scale graph processing on top of the state-of-the-art X10 programming language

Recently large-scale graphs with billions of vertices and edges have emerged in a variety of domains and disciplines especially in the forms of social networks, web link graphs, Internet topology graphs, etc. Mining these graphs to discover hidden knowledge requires particular middleware and software libraries that can harness the full potential of large-scale computing infrastructures such as super computers.

ScaleGraph is a graph library based on the highly productive X10 programming language. The goal of ScaleGraph is to provide large-scale graph analysis algorithms and efficient distributed computing framework for graph analysts and for algorithm developers, respectively.
ScaleGraph algorithms made Top #1 in Graph 500 benchmark.

Source: Dr. Toyotaro Suzumura, ICPE2014 keynote
Build an open source **Highly Scalable Large Scale Graph Analytics Library** beyond the scale of billions of vertices and edges on Distributed Systems.
Graph Algorithms

Currently supported algorithms

PageRank
Degree Distribution
Betweenness Centrality
Shortest path
Breadth First Search
Minimum spanning tree (forest)
Strongly connected component
Spectral clustering
Separation of Degree (HyperANF)
Cluster Coefficient

The algorithms that will be supported in the future.

Blondel clustering
Eigen solver for sparse matrix
Connected component
Random walk with restart
etc.
Weak Scaling and Strong Scaling Performance up to 128 nodes (1536 cores)

**Weak Scaling** Performance of Each Algorithm (seconds): RMAT Graph of Scale 22 per node

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PageRank</th>
<th>BFS</th>
<th>SSSP</th>
<th>WCC</th>
<th>SC</th>
<th>HyperANE</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMAT, Scale 22, 1 nodes</td>
<td>13.7</td>
<td>1.9</td>
<td>8.9</td>
<td>5.6</td>
<td>351.1</td>
<td>50.3</td>
<td>33.1</td>
</tr>
<tr>
<td>RMAT, Scale 26, 16 nodes</td>
<td>28.3</td>
<td>4.0</td>
<td>13.5</td>
<td>12.0</td>
<td>701.4</td>
<td>88.9</td>
<td>36.3</td>
</tr>
<tr>
<td>RMAT, Scale 28, 64 nodes</td>
<td>37.9</td>
<td>7.5</td>
<td>18.8</td>
<td>17.0</td>
<td>1166.0</td>
<td>103.5</td>
<td>39.4</td>
</tr>
<tr>
<td>RMAT, Scale 29, 128 nodes</td>
<td>45.3</td>
<td>11.2</td>
<td>24.5</td>
<td>22.1</td>
<td>1438.8</td>
<td>142.3</td>
<td>41.1</td>
</tr>
<tr>
<td>Random, Scale 29, 128 nodes</td>
<td>46.5</td>
<td>8.8</td>
<td>20.6</td>
<td>21.4</td>
<td>1106.6</td>
<td>162.3</td>
<td>42.7</td>
</tr>
</tbody>
</table>

**Strong Scaling** Performance of Each Algorithm (seconds): RMAT Graph of Scale 28

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PageRank</th>
<th>BFS</th>
<th>SSSP</th>
<th>WCC</th>
<th>SC</th>
<th>HyperANE</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 nodes</td>
<td>124.1</td>
<td>21.9</td>
<td>65.8</td>
<td>55.9</td>
<td>2969.9</td>
<td>38.0</td>
<td>16.1</td>
</tr>
<tr>
<td>32 nodes</td>
<td>91.7</td>
<td>18.7</td>
<td>36.9</td>
<td>30.2</td>
<td>1639.0</td>
<td>27.0</td>
<td>11.6</td>
</tr>
<tr>
<td>64 nodes</td>
<td>38.1</td>
<td>7.5</td>
<td>20.1</td>
<td>17.2</td>
<td>1169.9</td>
<td>10.6</td>
<td>4.9</td>
</tr>
<tr>
<td>128 nodes</td>
<td>26.5</td>
<td>5.8</td>
<td>14.7</td>
<td>10.5</td>
<td>706.4</td>
<td>6.8</td>
<td>3.1</td>
</tr>
</tbody>
</table>

Evaluation Environment: TSUBAME 2.5 (Each node is equipped with two Intel® Xeon® X5760 2.93 GHz CPUs by each CPU having 6 cores and 12 hardware threads, 54GB of memory. All compute nodes are connected with InifinitBand QDR)
Web Site and Source Code Repository

Official web site – http://scalegraph.org
  Project information
  Source code distribution
  Documentation
Source code repository - http://github.com/scalegraph/
  License: Eclipse Public License v1.0
  Project information and Documentation
ScaleGraph Software Stack

User Program

Graph Algorithm

User Program

File IO

BLAS for Sparse Matrix

X10

XPregel
Graph Processing System

Optimized Team

X10 Core Lib

ScaleGraph Core Lib

X10 Native Runtime

Third-Party Library Interface

Third-Party Libraries
(ARPACK, METIS)

X10 & C++

MPI
Developing Graph Algorithms (e.g. PageRank)

```
xpgraph.iterate[Double, Double](
    // Compute closure
    (ctx: VertexContext[Double, Double, Double, Double], messages: MemoryChunk[Double]) => {
        val value: Double;
        if (ctx.superstep() == 0) {
            // calculate initial page rank score of each vertex
            value = 1.0 / ctx.numberOfVertices();
        } else {
            // for step onward,
            value = (1.0 - damping) / ctx.numberOfVertices() +
                damping * MathAppend.sum(messages);  
        // sum score
        ctx.aggregate(Math.abs(value - ctx.value()));
        // set new rank score
        ctx.setValue(value);
        // broadcast its score to its neighbors
        ctx.sendMessageToAllNeighbors(value / ctx.outEdgesId().size());
    },

    // Aggregate closure: calculate aggregate value
    (values: MemoryChunk[Double]) => MathAppend.sum(values),

    // End closure: should continue?
    (superstep: Int, aggVal: Double) => {
        return (superstep >= maxIter || aggVal < eps);
    });

    public def iterate[M, A](
        compute: (ctx: VertexContext[V, E, M, A], messages: MemoryChunk[M]) => void,
        aggregator: (MemoryChunk[A]) => A,
        end: (Int, A) => Boolean)
```
Developing Graph Algorithms (e.g. PageRank)

The core algorithm of a graph kernel can be implemented by calling `iterate` method of XRegelGraph as shown in the example. Users are also required to specify the type of messages (M) as well as the type of aggregated value (V).

The method accepts three closures: `compute` closure, `aggregator` closure, and `end` closure.

In each superstep (iteration step), a vertex contributes its value, which depends on the number of links, to its neighbors.

Each vertex summarizes the score from its neighbors and then set the
Installation and Developing Graph Algorithms

Installation and Execution Guide

http://www.scalegraph.org/web/index.php/documentation/getting-started-guides

PageRank Example:

https://github.com/scalegraph/scalegraph/blob/develop/src/example/PageRankSimple.x10
Homework 3  (due Monday 11/14)

1. Graph Database:
   1.1: Download IBM System G Graph Tools.
   1.2: Download URL links in the Wikipedia data. Create a knowledge graph. Inject it into a graph database. Try some graph queries using Gremlin to find relevant terms. This can serve as keyword expansion. Show some visualizations of your queries.

2. Graph Topology Analysis:
   Choose a subgraph of the graph in 1. or another graph that you want to do:
   2.1: Calculate the below Centralities:
       - In-degree, out-degree
       - Betweenness
       - PageRank
   2.2: Choose some points, calculate the shortest path between them.

3. Bayesian Network:
   Use an example dataset to be provided by TA or choose your own dataset
   3.1: Define the Bayesian Network structure
   3.2: Run System G Bayesian Network tool to convert the structure into Junction Tree
   3.3: Test some examples to show how Bayesian Network works.
Questions?