TigerGraph | Graph Gurus 33

GSQL Writing Best Practices

Part 2 - A Better Plan

Today's Host

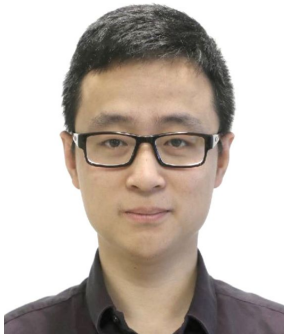
David Ronald

Director of Product Marketing

- BSc in Applied Physics from Strathclyde University, MSc in Optoelectronic & Laser Devices from St Andrews
- Prior work in artificial intelligence, natural linguistic programming and telecommunications technology
- 18+ years in tech industry

TigerGraph

Today's Presenter



Xinyu Chang

Director of Customer Solutions

- Co-authored GSQL, TigerGraph's query language, and expertise in graph solutions and algorithms
- Developed solutions for many Fortune 50 companies
- Over 5 years with TigerGraph

TigerGraph

Some Housekeeping Items

- Although your phone is muted we do want to answer your questions - **submit your questions at any time** using the Q&A tab in the menu

- The webinar is being recorded and will uploaded to our website shortly (https://www.tigergraph.com/webinars/) and the URL will be emailed you

- If you have issues with Zoom please contact the panelists via chat

TigerGraph

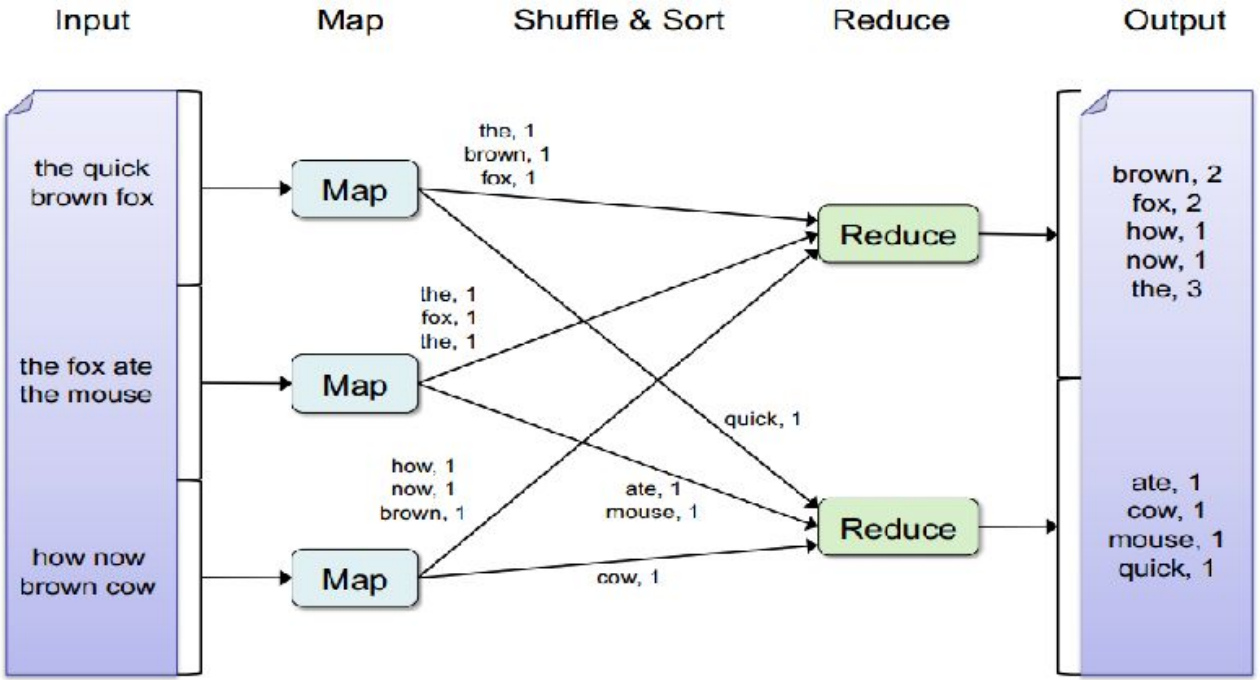
Thinking in GSQL - Agenda

1. Review the Basics II
2. What is a Better Plan?
3. How to Check the Log
4. Example 1
5. Example 2
6. Example 3

The Basics II

The Basics II

map-reduce



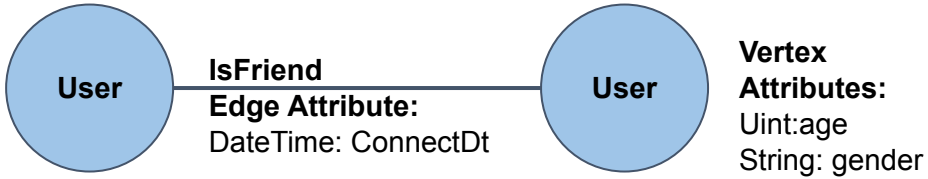
TigerGraph

The Basics II

1. ACCUM runs on the edges(vertices if not edges referred).
2. POST-ACCUM runs on the vertexes after ACCUM.
3. Both ACCUM and POST-ACCUM are running logic of **map**
4. After execution of ACCUM and POST-ACCUM there is a hidden **reduce** phase is executed to aggregate the values.

Exampe Setup

In this tutorial the example will be given based on schema below. This is a social network of people who are connected as friends.

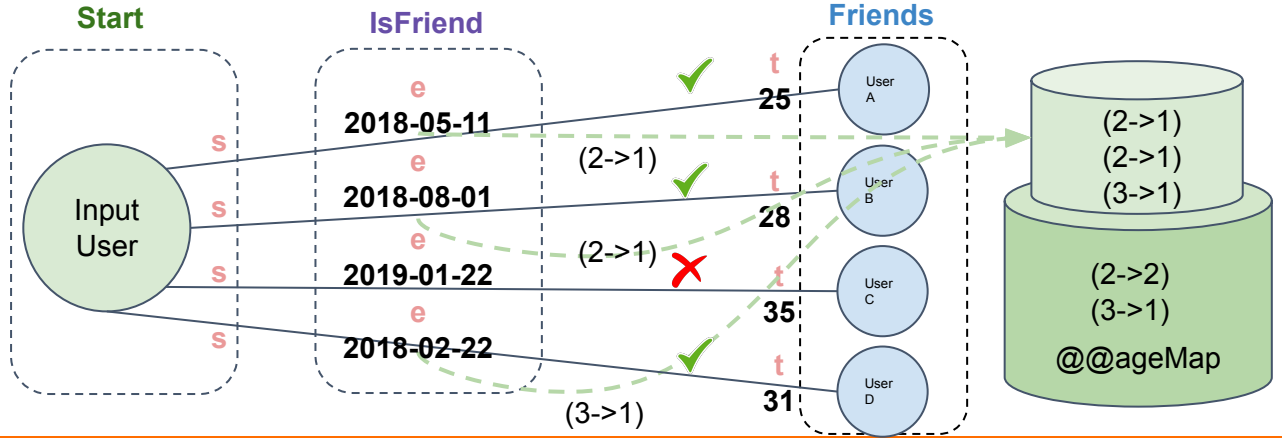


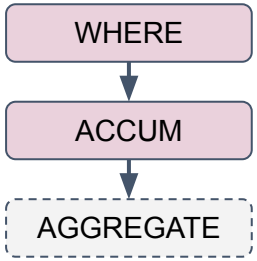
TigerGraph

ACCUM Clause

What is the age distribution of friends that were registered in 2018?

```
CREATE QUERY GetFriends(vertex<User> inputUser) FOR GRAPH Social {
    MapAccum<uint, SumAccum<uint>> @@ageMap;
    Start = {inputUser};
    Friends = SELECT t FROM Start:s-(IsFriend:e)-:t
              WHERE e.connectDt BETWEEN to_datetime("2018-01-01")
                AND to_datetime("2019-01-01")
              ACCUM @@ageMap += (t.age/10->1);
    PRINT @@ageMap;
}
```





Select the matching edges

Local compute + send messages

Aggregate the messages to accumulator

- Each edge satisfying the FROM & WHERE clauses performs the **ACCUM** clause statements.
- **ACCUM** has access to **s**, **e** and **t.**
- In **ACCUM,** vertices do not see each other's updates b/c updates aren't processed until the **AGGREGATE** step.
- The **AGGREGATE** phase is done automatically after **ACCUM**. After that, the updated accumulator value can be accessed

- **+=** means sending message to accumulator

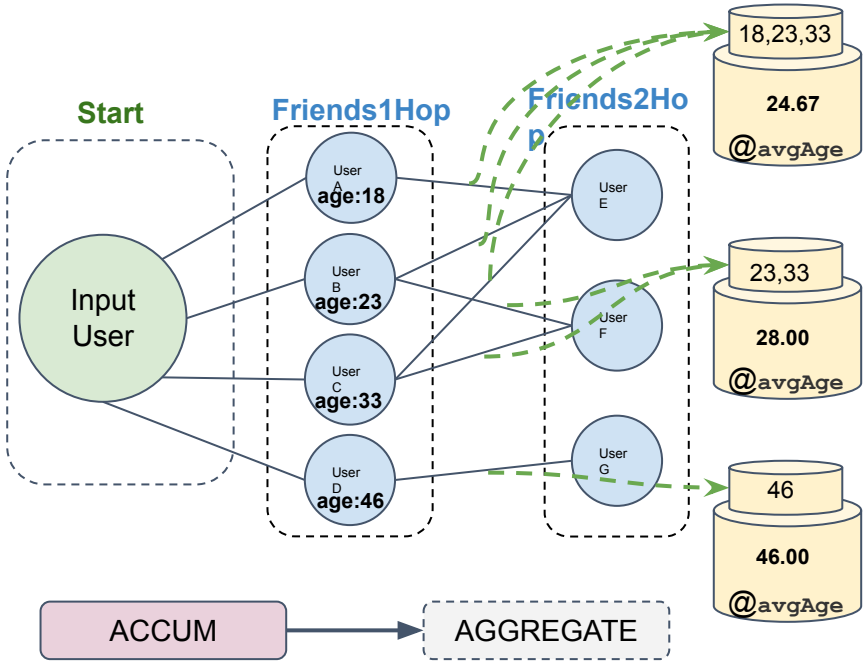
TigerGraph

ACCUM Clause

Given an input user. Output the average age of their common friends.

```
CREATE QUERY GetFriends(vertex<User> inputUser) FOR
GRAPH Social {
    AvgAccum @avgAge;
    Start = {inputUser};
    Friends1Hop = SELECT t FROM Start:s-(IsFriend:e)-:t;
    Friends2Hop = SELECT t
                  FROM Friends1Hop:s-(IsFriend:e)-:t
                  ACCUM t.@avgAge += s.age;
    print Friends2Hop;
}
```

- Update of local accumulator cannot be seen during **ACCUM** phase
- The messages will be aggregated during **AGGREGATE** phase based on accumulator type.



Send the messages to target nodes

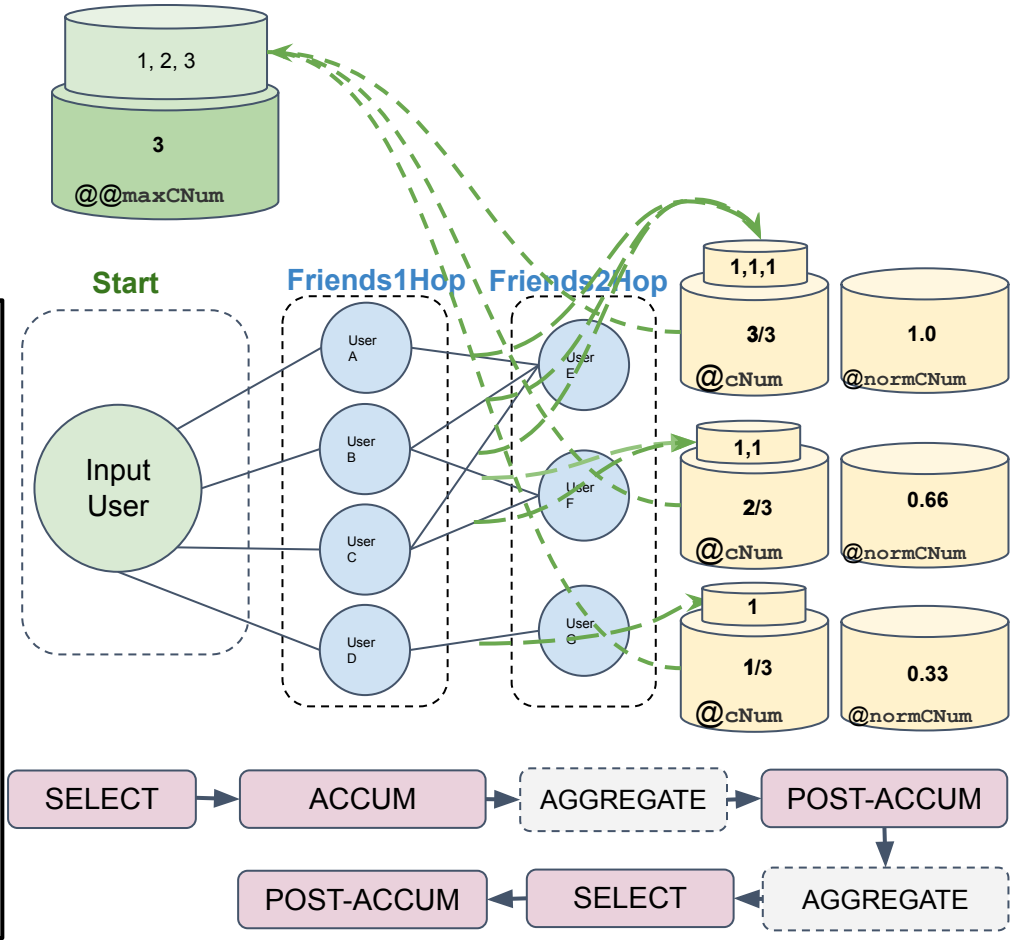
Aggregate the messages to accumulator

TigerGraph

POST-ACCUM

Given a set of persons (friends of friends of the inputUser), output the normalized number of common friends for each person in the set.

```
1 CREATE QUERY GetFriends(vertex<User> inputUser) FOR
GRAPH Social {
2     SumAccum<uint> @cNum;
3     SumAccum<float> @normCNum;
4     MaxAccum<float> @@maxCNum;
5     Start = {inputUser};
6     Friends1Hop = SELECT t FROM Start:s-(IsFriend:e)-:t;
7     Friends2Hop = SELECT t
8                   FROM Friends1Hop:s-(IsFriend:e)-:t
9                   ACCUM t.@cNum += 1
10                  POST-ACCUM @@maxCNum += t.@cNum;
11    Friends2Hop = select s FROM Friends2Hop:s
12                  POST-ACCUM
13                  s.@normCNum = s.@cNum/@@maxCNum;
14    print Friends2Hop;
15 }
```



TigerGraph

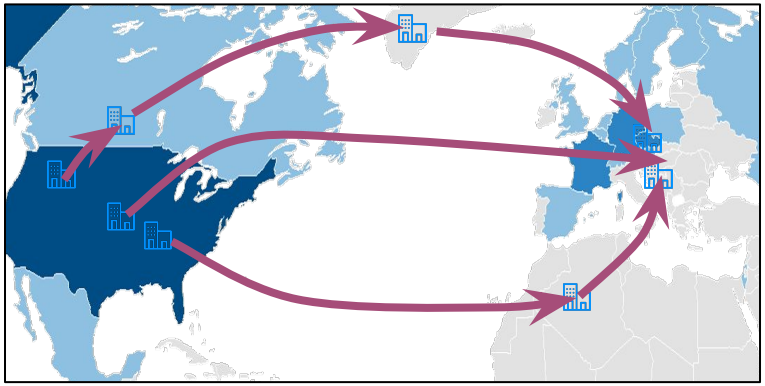
A Better Traversal Plan

1. Design the lightest weight traversal path
2. Think twice before starting a query with all vertices (of a given type)
3. Make the algorithm bidirectional
4. Avoid hub nodes, do the moonwalk
5. Multiple search conditions

A Better Traversal Plan

1. Design the lightest weight traversal plan

Similar to relational DB query optimization, start with smaller sets, and prune your sets as early as possible.

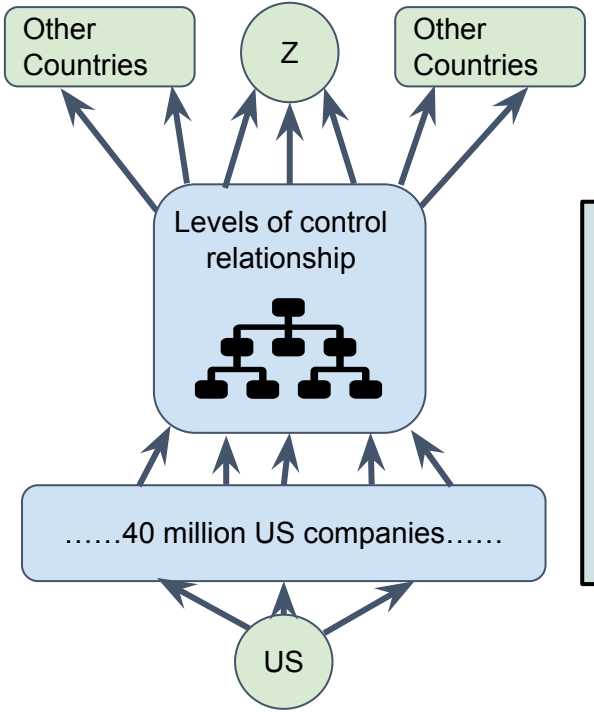


Example: Find the US domiciled companies that have ultimate parent company in country Z. Z has fewer companies than the US.

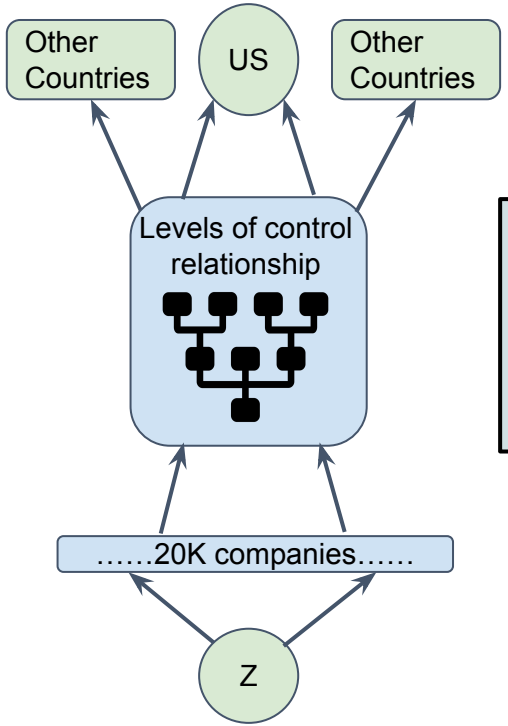
By starting with the smaller set (Z instead of US), you can reduce the amount of computation.

A Better Traversal Plan

1. Design the lightest weight traversal plan



- Starting from US requires finding ultimate parent companies for all 40 million US companies.
- The total number of edges traversed is in the 100s of millions



- Starting from Z, we only need to start with about 20K companies
- The total number of edges traversed is below 1 million

TigerGraph

A Better Traversal Plan

2. Think twice before starting a query with all vertices (of a given type).

Start = {TYPEA.*}

Start = {ANY};

Is it possible to start from a small set of vertex IDs?

Only start from an entire vertex type when you have to.

```
CREATE QUERY q1 (vertex v) FOR
GRAPH g1 {
  Start = {company.*};
  Start = SELECT s FROM Start:s
  WHERE s == v;
  ...
}
```

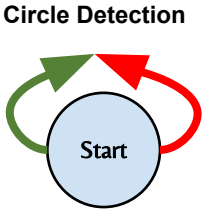
```
CREATE QUERY q1 (vertex<company>
v) FOR GRAPH g1 {
  Start = {v};
  ...
}
```

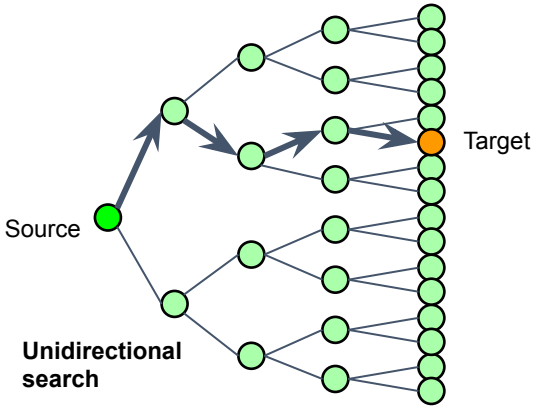
TigerGraph

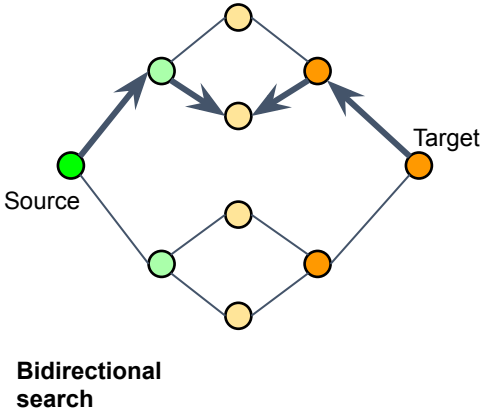
A Better Traversal Plan

3. Start the traversal bidirectionally

When trying to find a path, it is much faster to do the traversal bidirectionally. For example: shortest path query and circle detection query.







Why? Because the number of edges traversed is reduced exponentially.

Suppose each vertex has an average of 10 edges, and shortest path from Source to Target turns out to be 4 hops. With a unidirectional search, we will traverse 10*10*10*10 = 10,000 edges. With bidirectional search, we will traverse only 10*10 + 10*10 = 200 edges.

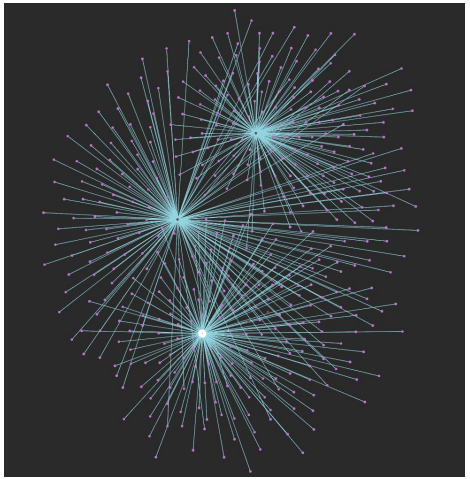
TigerGraph

A Better Traversal Plan

4. Avoid hub nodes

Hub Nodes or **Super Nodes** are vertices having a huge number of neighbors. When traversal encounters such nodes it has to touch a very large portion of the graph, which hinders the query.

Design the traversal plan to avoid starting from the hub nodes.

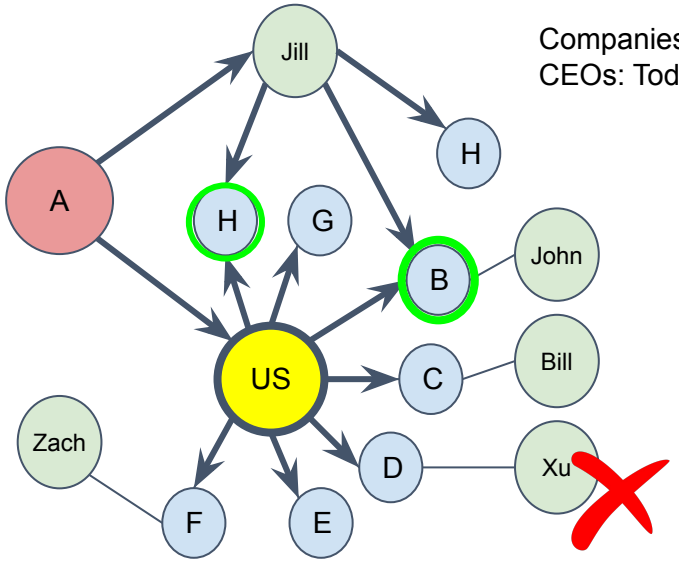


TigerGraph

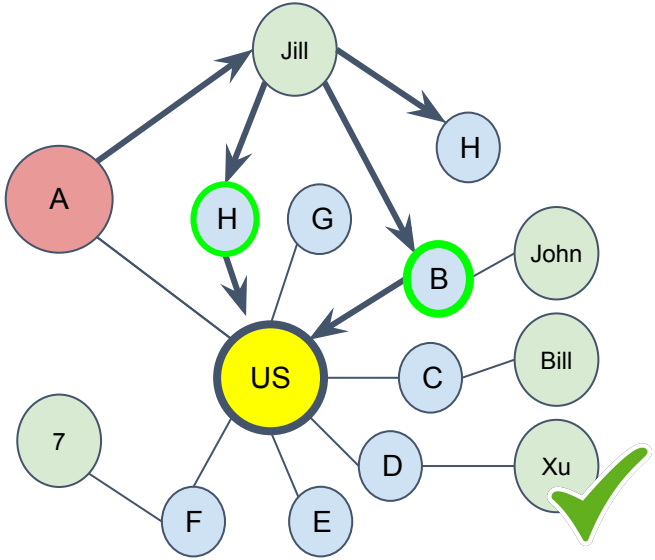
A Better Traversal Plan

4. Avoid hub nodes

Example: Given a company **A**, find all companies that are in the same country and were ran by the same CEO



Companies: A,B,C,...
CEOs: Todd, John, Bill...

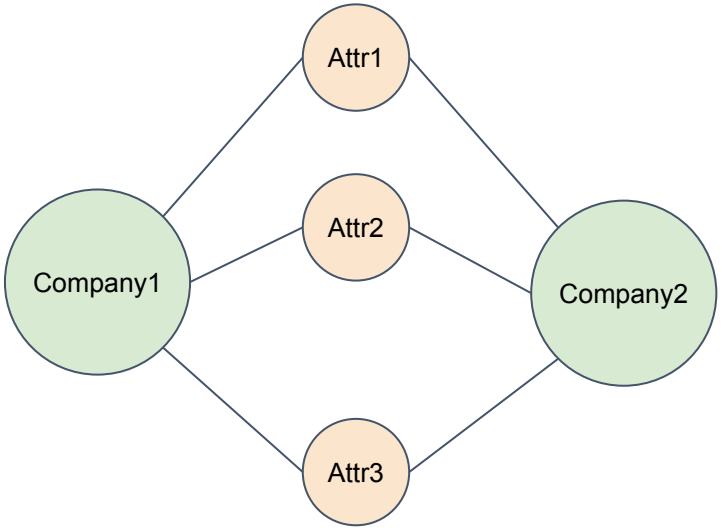


TigerGraph

A Better Traversal Plan

4. Avoid hub nodes

When there are multiple searching conditions. Start with the most special one.



Vertex/EdgeType	Count
Attr1	10
Attr2	1,000,000
Attr3	10,0000
Attr1_Company	100,000,000
Attr2_Company	100,000,000
Attr3_Company	100,000,000

TigerGraph

A Better Traversal Plan

4. Avoid hub nodes

Alternatively, when an approximated result is good enough, you can also consider filtering the hub nodes out in your WHERE clause. Or use the SAMPLE clause to sample a subset of the neighbors.

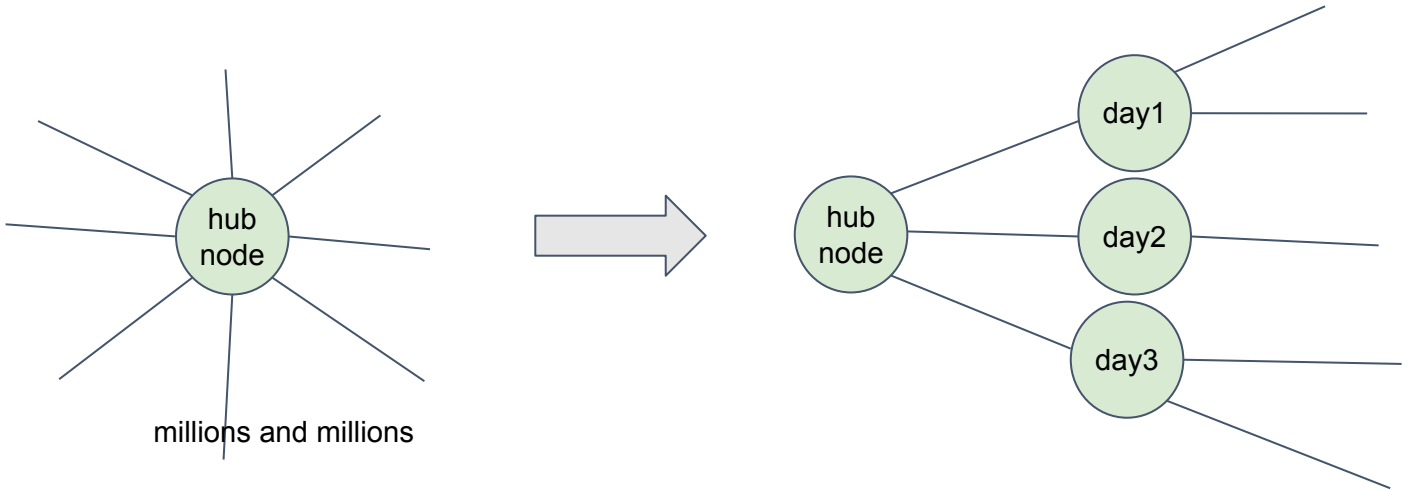
WHERE t.outdegree() < 100000

SAMPLE 100 **EDGE WHEN** s.outdegree() > 1000000

A Better Traversal Plan

4. Avoid hub nodes

Split the hub node at schema level.

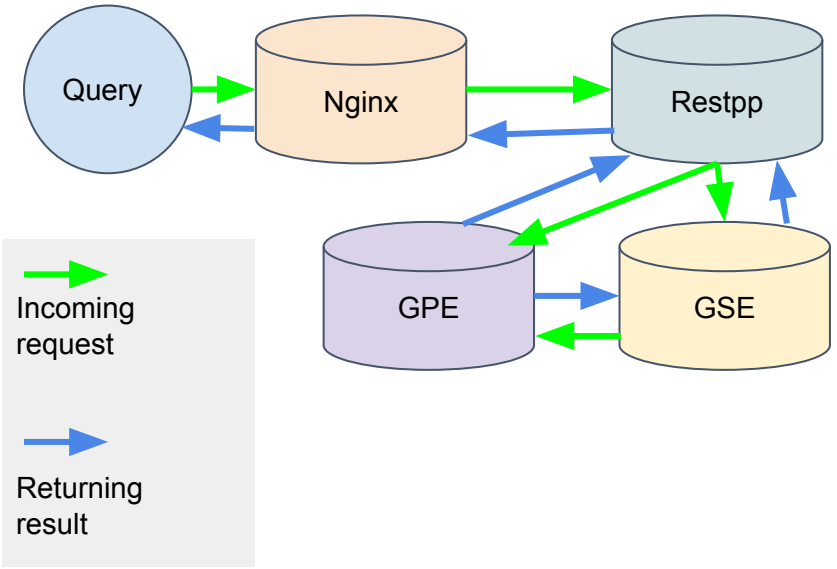


TigerGraph

How to Check Query Log

Query Problems -- Log Checking

Query Execution Flow Chart



1. Nginx receives request
2. Nginx send request to Restpp
3. Restpp send ID translation task to GSE, and query request to GPE
4. GSE send translated ID to GPE, GPE starts to process query
5. GPE send result to restpp, GPE send translation task to GSE, GSE send translation result to Restpp
6. Restpp send result back to Nginx
7. Nginx send the response

TigerGraph

Query Problems -- Log Checking

Nginx receives the request

```
>grep QUERY_NAME ~/tigergraph/logs/nginx/nginx_1.access.log
```

```
tigergraph@ubuntu:~/tigergraph/logs/nginx$ grep InvitedUserBehavior  nginx_1.access.log
127.0.0.1 - - [21/Feb/2019:15:11:42 -0800] "GET /engine/query/AntiFraud/InvitedUserBehavior?inputUser=11 HTTP/1.1" 202 67 "http
://localhost:14240/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.86 Safari/537.36"
```

Nginx send request to Restpp

```
>grep QUERY_NAME /home/tigergraph/tigergraph/logs/RESTPP_1_1/log.INFO
```

```
grep InvitedUserBehavior /home/tigergraph/tigergraph/logs/RESTPP_1_1/log.INFO
I0221 15:11:42.138013  9181 handler.cpp:235] Engine_req|RawRequest|196610:RESTPP_1_1:1550790702138|GET|url = /query/AntiFraud/InvitedUserBehavior?inputUser
=11&|payload_data.size() = 2|api = v2
```

Request ID

Note:
Here **1550790702138** is the request ID. With request ID all logs in Restpp, GPE and GSE can be found.

TigerGraph

Query Problems -- Log Checking

GPE Process Query

GPE log is very important, most of time of a query is spent in GPE, GPE log gives you the detailed info of query Execution. Such as data amount has been processed, time elapsed in each ACCUM and POST-ACCUM clause.

```
>grep REQUEST_ID /home/tigergraph/tigergraph/logs/GPE_1_1/log.INFO
```

```
tigergraph@ubuntu:~/tigergraph/logs/nginx$ grep 1550790702138 /home/tigergraph/tigergraph/logs/GPE_1_1/log.INFO
I0221 15:11:42.142649 10265 serviceapi.cpp:563] Request|default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|Start_RunUDF
I0221 15:11:42.142686 10265 gtimer.cpp:134] (0.000 ms) |default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|UDF::Start
I0221 15:11:42.142748 10265 gtimer.cpp:134] (0.066 ms) |default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|UDF::Initialization
I0221 15:11:42.142918 10265 gtimer.cpp:134] (0.171 ms) |default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|UDF::Finished iteration 1 which did 2 edge maps a
nd 0 vertex maps (in 0.104 ms), 2 vertex reduces (in 0.058 ms), and activated 2 vertices.
I0221 15:11:42.143066 10265 gtimer.cpp:134] (0.144 ms) |default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|UDF::Finished iteration 2 which did 1 edge maps a
nd 0 vertex maps (in 0.104 ms), 1 vertex reduces (in 0.031 ms), and activated 1 vertices
I0221 15:11:42.143324 10265 gtimer.cpp:134] (0.257 ms) |default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|UDF::Finished iteration 3 which did 5 edge maps a
nd 0 vertex maps (in 0.049 ms), and activated 5 vertices.
I0221 15:11:42.143454 10265 gtimer.cpp:134] (0.730 ms) |default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|UDF::End
I0221 15:11:42.143566] Request|default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|Stop_RunUDF|0ms
I0221 15:11:42.143529 10265 enginejobrunner.cpp:320] Request|default,196610:RESTPP_1_1:1550790702138:NYN,16,,0|Finished in 4.218 ms Response 1277 bytes Id
conversion 9
```

Query Started

End of execution

Query execution time

Number of ACCUM clauses executed in each SELECT statement

Time spent in ACCUM clause

Number of POST-ACCUM clauses executed in each SELECT statement and its corresponding execution time

# of vertices that has been selected to the vertex set

SELECT statements executed

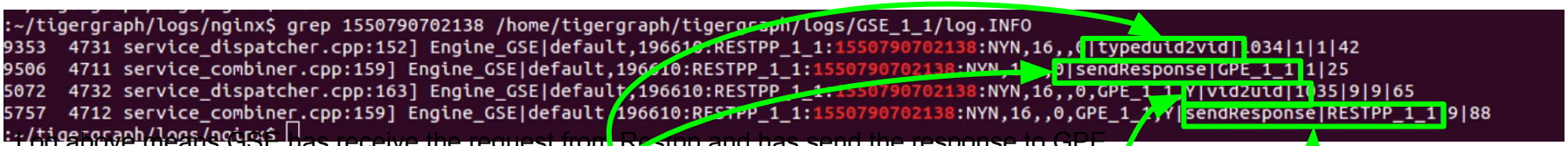
Sent task to GSE and response to Restpp

TigerGraph

Query Problems -- Log Checking

GSE Process ID Translation Tasks

```
>grep REQUEST_ID /home/tigergraph/tigergraph/logs/GSE_1_1/log.INFO
```



Log above means GSE has receive the request from Restpp and has send the response to GPE

Note:
Here **uid** means external string ID, and **vid** means internal vertex ID.

uid2vid request from Restpp

Send response to GPE

vid2uid request from GPE

Send response to Restpp

TigerGraph

Query Problems -- Log Checking

Restpp return the result to Nginx

>grep REQUEST_ID /home/tigergraph/tigergraph/logs/RESTPP_1_1/log.INFO

```
tigergraph@ubuntu:~/tigergraph/logs/nginx$ grep 1550790702138 /home/tigergraph/tigergraph/logs/RESTPP_1_1/log.INFO
I0221 15:11:42.138013  9181 handler.cpp:235] Engine_req|RawRequest|196610:RESTPP_1_1:1550790702138|GET|url = /query/AntiFraud/InvitedUserBehavior?inputUser
=11&|payload_data.size() = 2|api = v2
I0221 15:11:42.146179  9182 requestrecord.cpp:221] Engine_req|ReturnResult|196610:RESTPP_1_1:1550790702138|1167
```

Return to Nginx

Nginx send out the response

>grep QUERY_NAME ~/tigergraph/logs/nginx/nginx_1.access.log

```
tigergraph@ubuntu:~/tigergraph/logs/nginx$ grep InvitedUserBehavior  nginx_1.access.log
127.0.0.1 - - [21/Feb/2019:15:11:42 -0800] "GET /engine/query/AntiFraud/InvitedUserBehavior?inputUser=11 HTTP/1.1" 202 67 "http
://localhost:14240/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.86 Safari/537.36"
127.0.0.1 - - [21/Feb/2019:15:11:42 -0800] "GET /query/AntiFraud/InvitedUserBehavior?inputUser=11 HTTP/1.1" 200 1167 "-" "-"
```

The Thinking Steps

The Thinking Steps

1. Design a traversal plan
   Where to start from? What are the steps? What edge to use for each step?

2. Choose and define the accumulators
   What needs to be in the result? Where is the info needed? What accumulator to use?

3. Populate the accumulators
   How do we gather the info to the right place?

4. Print the result

Examples 1

Find the US domiciled companies that have ultimate parent company in country Z. Z has fewer companies than the US.

The Thinking Steps

1. Design a traversal plan

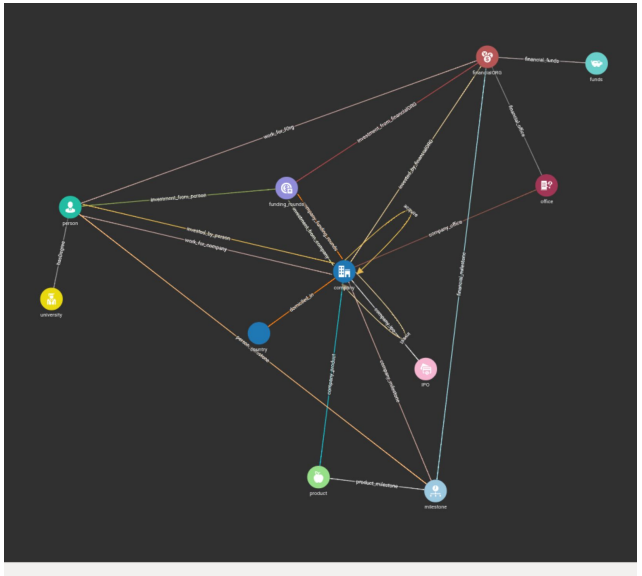
   Where to start from? **US, Z**

   What are the steps?

   Option 1: US->company->*company->Z
   Option 2: Z->company->*company->US

   What edge to use for each step?

   domiciled, invest/acquire, domiciled



TigerGraph

The Thinking Steps

2. Choose and define the accumulators

What is the final result?

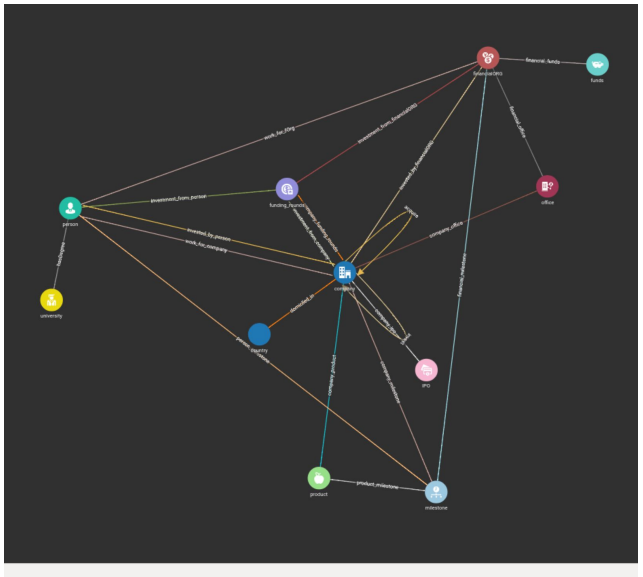
The list of company

Where is the info needed for the final result?

Which vertex is the ultimate parent company
Which company is domiciled in US
Which company is domiciled in Z
A company invest/acquire another company

What accumulator to use?

ListAccum<vertex>



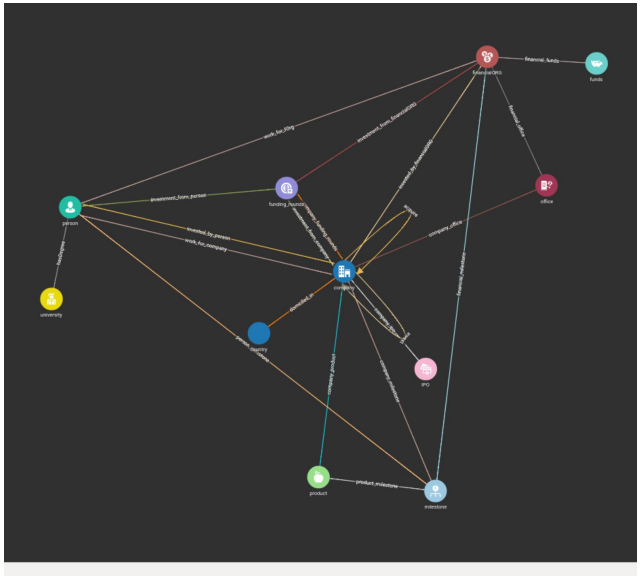
TigerGraph

The Thinking Steps

3. Populate the accumulators

How do we gather the info to the right place?

Pass the dates of claims of the input prescriber to the patient vertex



TigerGraph

Examples 2

Given a company A, find all companies that are in the same country and were ran by the same CEO.

The Thinking Steps

1. Design a traversal plan

   Where to start from? **Company A**

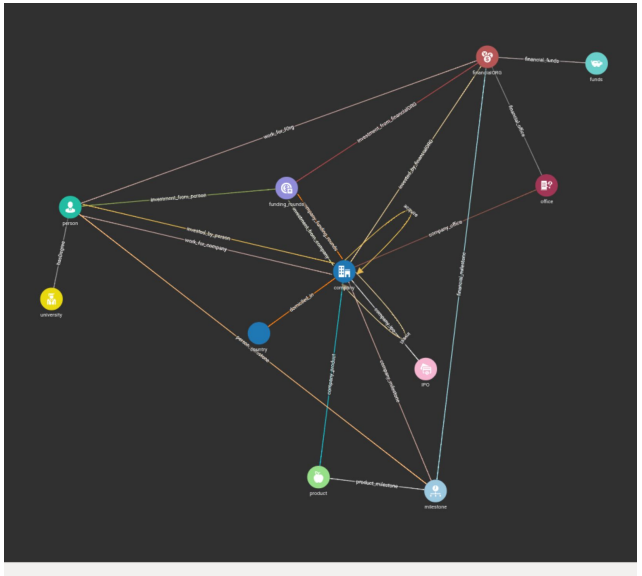
   What are the steps?

   Option 1 A -> employee -> company
   A -> country -> company

   Option 2 A -> country
   A -> employee -> company -> country

   What edge to use for each step?

   work_for_company, domiciled_in



TigerGraph

The Thinking Steps

2. Choose and define the accumulators

What is the final result?

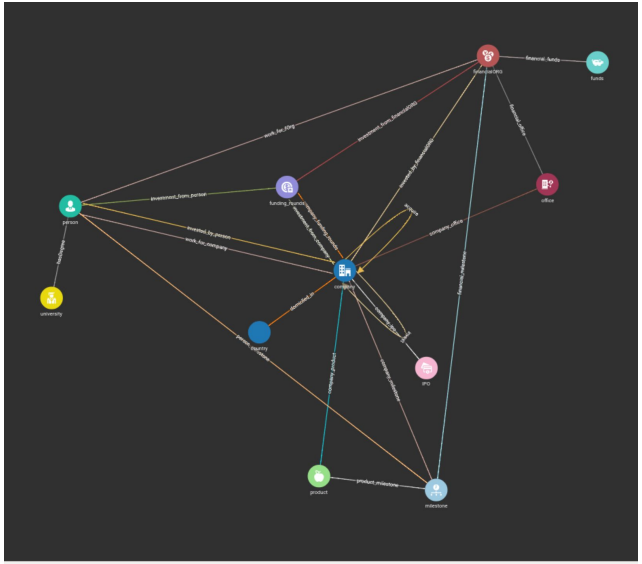
A list of company

Where is the info needed for the final result?

country, employee

What accumulator to use?

Option 1 : none
Option 2: OrAccum<BOOL>



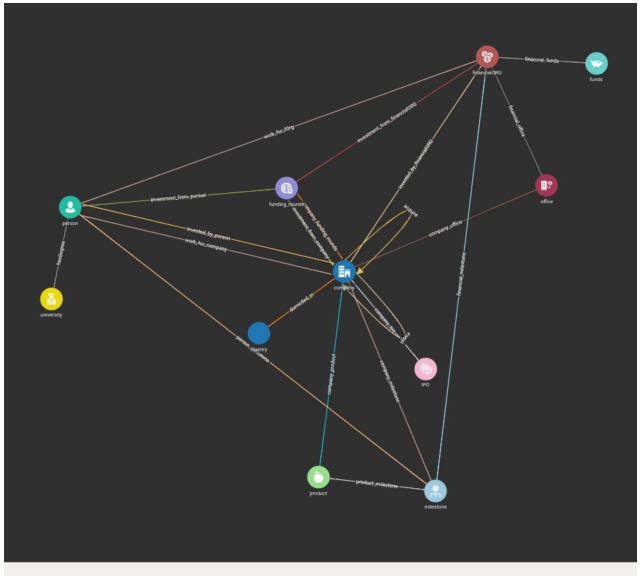
TigerGraph

The Thinking Steps

3. Populate the accumulators

How do we gather the info to the right place?

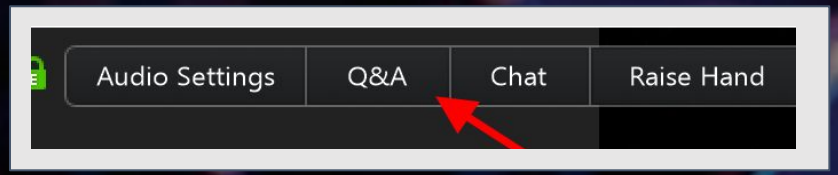
Mark the country of company A



TigerGraph

Q&A

Please submit your questions via the Q&A tab in Zoom



More Questions?

Join our Developer Forum
https://community.tigergraph.com

Join our Developer Chat
https://discord.gg/F2c9b9v

Sign up for our Developer Office Hours (Thursday at 11 AM PDT)
https://info.tigergraph.com/officehours

Additional Resources

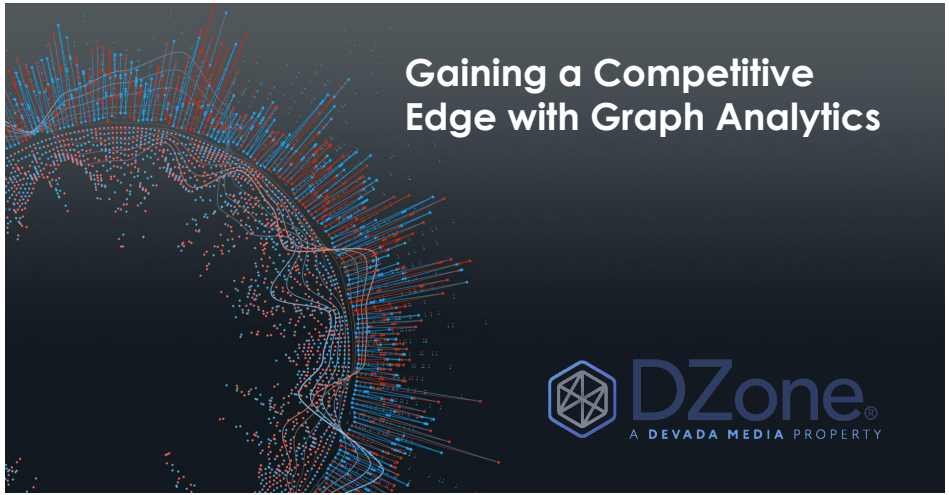
Start Free at TigerGraph Cloud
https://www.tigergraph.com/cloud/

Test Drive Online Demo
https://www.tigergraph.com/demo

Download the Developer Edition
https://www.tigergraph.com/download/

Guru Scripts
https://github.com/tigergraph/ecosys/tree/master/guru_scripts

Upcoming Webinars



Gaining a Competitive Edge with Graph Analytics

Wednesday, April 15, at 10am PDT

https://dzone.com/webinars/gain-a-competitive-edge-with-graph-analytics

TigerGraph

Thank You