CSE487/587: DATA INTENSIVE COMPUTING

Instructor: Bina Ramamurthy

Most Followed User on GitHub

Submitted By:

Aman Bhayana
50290968
Pratik Agarwal
50290570

Most Followed User on GitHub

Abstract:

GitHub is a web-based hosting service for version control using Git. It serves as a platform for hosting codes which provides Source Code Management as well as Distributed Version Control mechanism. Different users have their own programming preferences in terms of style, language, framework etc. There are many open-source as well as organizational level projects hosted on GitHub. There are many users who have many followers, whose projects have been forked the most etc. In short, this information can be used to determine as to which user has most influence on other GitHub users in terms of project forks, following. We aim to demonstrate this feature using Apache SPARK. The user data i.e. the followers of a given user can be procured using GitHub API or using the public dataset made available by GitHub on Google Big Query. The user details on this dataset can act as the nodes of a directed graph which would be our input. This input when visualized in form of a directed graph, can be simulated to obtain the result using the concept of Connected Components. It is often used in the most real-life social networking models. We use Apache SPARK as it uses Lazy Computation. It means that the transformations are not processed until some action is encountered or is encountered. It uses RDD as the lowest level of abstraction and data frames or datasets are based on RDD only. We can use any data visualization tool like Tableau or d3.js to demonstrate the results as per the problem proposition.

Problem Statement:

Estimation of the most followed person or a person who has the most impact on any website can be a tedious job if you have lots of data pertaining to that person. To be able to perceive that data and get meaningful conclusion out of it would require lots and lots of processing. However, the problem to determine the most followed person still remains. The problem presented can be solved using the libraries provided by Apache SPARK on any programming language which supports it. The solution we propose to this problem incorporates the concepts of connected components. A graph this size with million nodes would require a lot of computation and storage. A map-reduce algorithm on a distributed system utilizing the full functionalities of Apache SPARK would be apt. Apache SPARK provides RDDs which then reduces the amount of work and processing time. Using Apache SPARK provides an elegant solution to the problem and has its advantages of using a regular map reduce algorithm. We propose an elegant solution of using a Map-Reduce algorithm while using the functionalities provided by Apache SPARK in amalgamation with Connected Components. This solution can widely improve upon the computation time and storage issues. A distributed system architecture can widely support this solution.

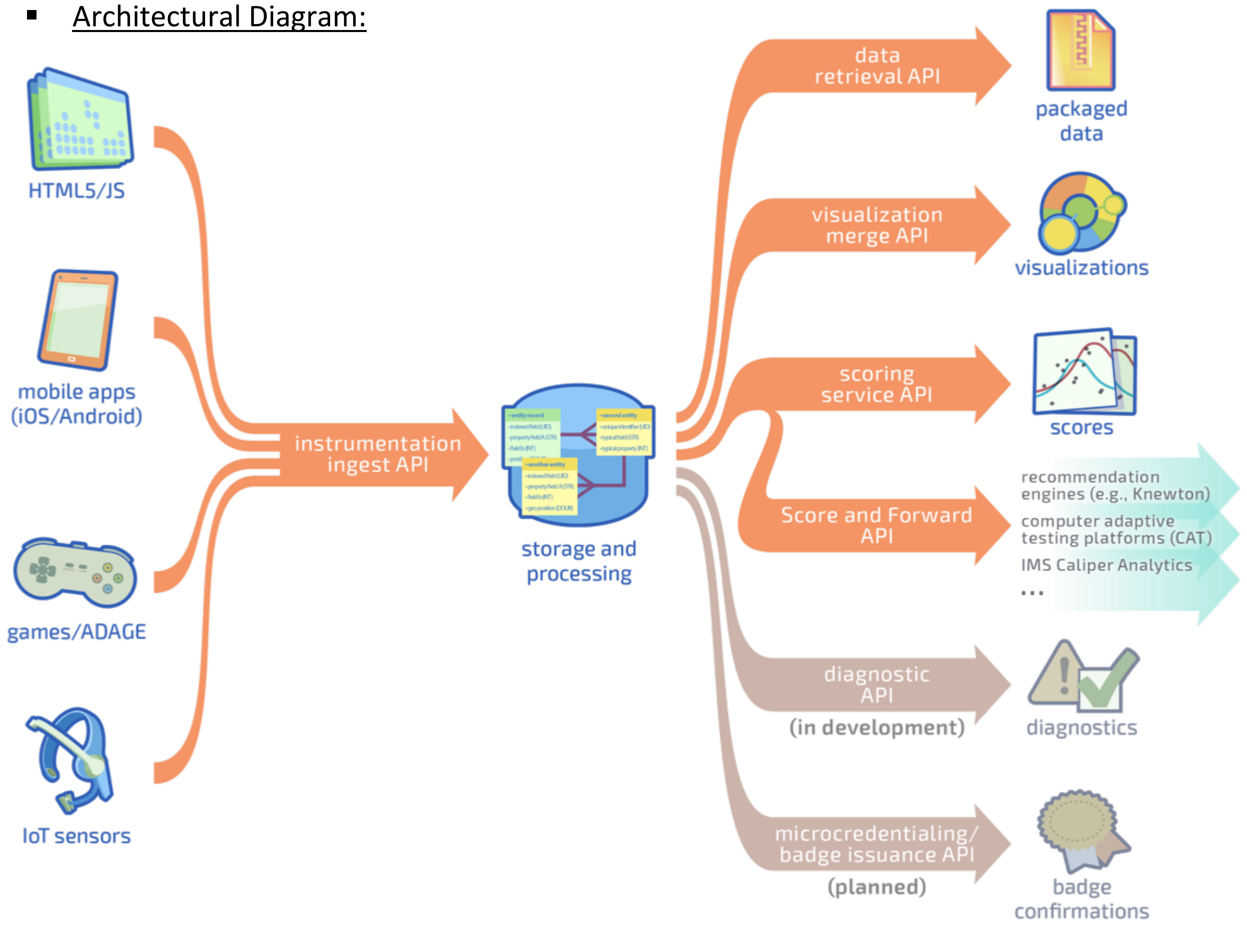
Solution and Design Document:

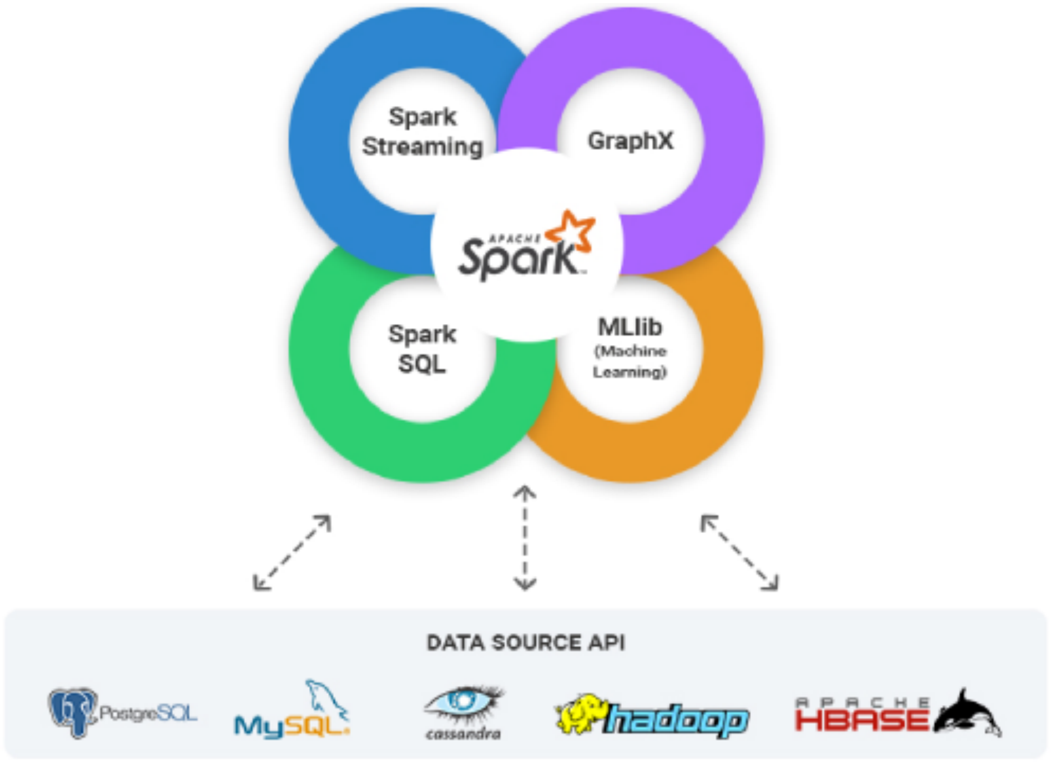
SPARK is a general-purpose distributed data processing engine that is suitable for use in wide range of circumstances. It is a fast in-memory data processing engine with elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that requires fast iterative access to datasets. We propose this solution using Apache SPARK as the level of abstraction it provides to access the datasets and the value it adds to the data science and visualization. The use of connected components concept provides the algorithm which would be driving force behind this implementation. The core idea behind this algorithm used is to first compute the similarity scores between nodes and then compute connected components subject to different similarity thresholds. This algorithm works on the principle of growing a Breadth First Tree from each node in parallel.

- ### Methodology:

  The data procured from GitHub will act as the inputs of a directed graph and then be fed to a Map-Reduce algorithm running on top of Apache SPARK. The final solution of this implementation would result in gathering user information who has the most followers. The algorithm can be tweaked to determine the top 10 users with most followers for the purpose of data visualization.

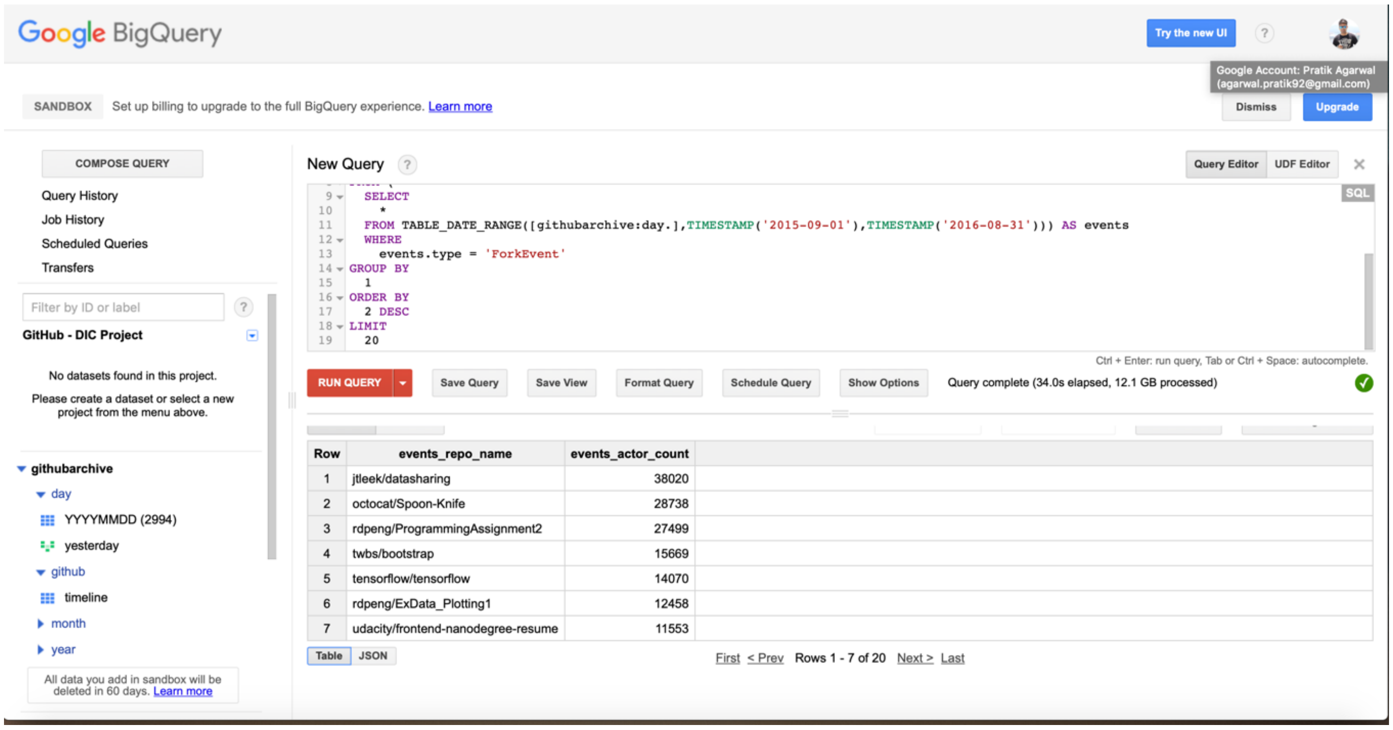
- ### Architectural Diagram:





- <u>Data Pipeline and Algorithm:</u>

  The entire GH Archive is also available as a public dataset on [Google BigQuery](): the dataset is automatically updated every hour and enables you to run [arbitrary SQL-like queries]() over the entire dataset in seconds. A screenshot data collection procedure can be seen below:


Google BigQuery
Try the new UI
Google Account: Pratik Agarwal (agarwal.pratik92@gmail.com)
SANDBOX
Set up billing to upgrade to the full BigQuery experience. Learn more
Dismiss
Upgrade
COMPOSE QUERY
Query History
Job History
Scheduled Queries
Transfers
Filter by ID or label
GitHub - DIC Project
No datasets found in this project.
Please create a dataset or select a new project from the menu above.
githubarchive
day
YYYYMMDD (2994)
yesterday
github
timeline
month
year
All data you add in sandbox will be deleted in 60 days. Learn more
New Query
Query Editor
UDF Editor
SQL
RUN QUERY
Save Query
Save View
Format Query
Schedule Query
Show Options
Table
JSON
First < Prev Rows 1 - 7 of 20 Next > Last

```sql
  SELECT
    *
  FROM TABLE_DATE_RANGE([githubarchive:day.],TIMESTAMP('2015-09-01'),TIMESTAMP('2016-08-31'))) AS events
  WHERE
    events.type = 'ForkEvent'
GROUP BY
  1
ORDER BY
  2 DESC
LIMIT
  20
```

Ctrl + Enter: run query, Tab or Ctrl + Space: autocomplete.

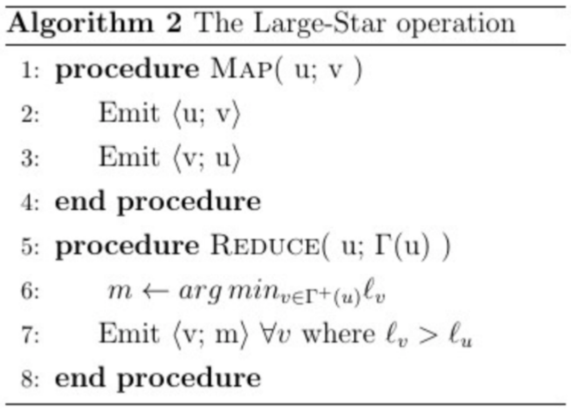
Query complete (34.0s elapsed, 12.1 GB processed)

Row	events_repo_name	events_actor_count
1	jtleek/datasharing	38020
2	octocat/Spoon-Knife	28738
3	rdpeng/ProgrammingAssignment2	27499
4	twbs/bootstrap	15669
5	tensorflow/tensorflow	14070
6	rdpeng/ExData_Plotting1	12458
7	udacity/frontend-nanodegree-resume	11553

The data obtained from the above tables can be formulated in the form of a graph in which the directed graph edges can act as input to the Map-Reduce Algorithm. The operation we propose to implement as follows:

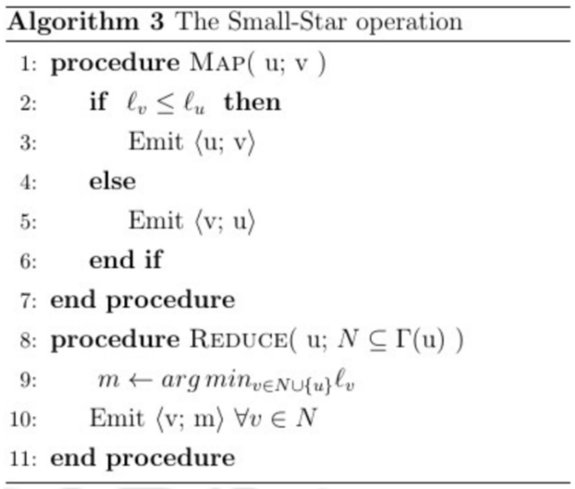
i. There are two operations we will be using: Large Star Operation and Small Star Operation. Large Star: Connect all strictly larger neighbors to the minimum neighbor including self. Star Star: Connect all smaller neighbors and self to minimum neighbor.

   The algorithmic implications can be seen as below:



Algorithm 2 The Large-Star operation

```
procedure MAP( u; v )
    Emit ⟨u; v⟩
    Emit ⟨v; u⟩
end procedure
procedure REDUCE( u; $\Gamma$(u) )
    $m \leftarrow arg\,min_{v \in \Gamma^{+}(u)} \ell_v$
    Emit ⟨v; m⟩ $\forall v$ where $\ell_v > \ell_u$
end procedure
```



Algorithm 3 The Small-Star operation

```
procedure MAP( u; v )
    if $\ell_v \leq \ell_u$ then
        Emit ⟨u; v⟩
    else
        Emit ⟨v; u⟩
    end if
end procedure
procedure REDUCE( u; $N \subseteq \Gamma$(u) )
    $m \leftarrow arg\,min_{v \in N \cup \{u\}} \ell_v$
    Emit ⟨v; m⟩ $\forall v \in N$
end procedure
```

ii. The algorithm which we propose to incorporate the above two operations is called "*Two Phase Algorithm*". It uses both the large star operation and small star operation until convergence. It is defined as follows:

   Pseudo Code:

```
Input: Edges (u, v) as a set of key-value pairs <u; v>.
Input: A unique label lv for every node v ∈ V .
repeat
    repeat
        Large-Star
    until Convergence
    Small-Star
until Convergence
```

   The algorithm processes a simple map reduce solution which gets the input as directed edges of a graph and performs the large star and small star operation as per the above implementation and finally determines the node with most convergence, i.e. as per our problem the user represented by that node will be the one with most followers.

   We went ahead and formulated a basic framework of the code which would perform the above operation using the extensions provide by Apache SPARK. This implementation makes use of RDDs and is coded in python.

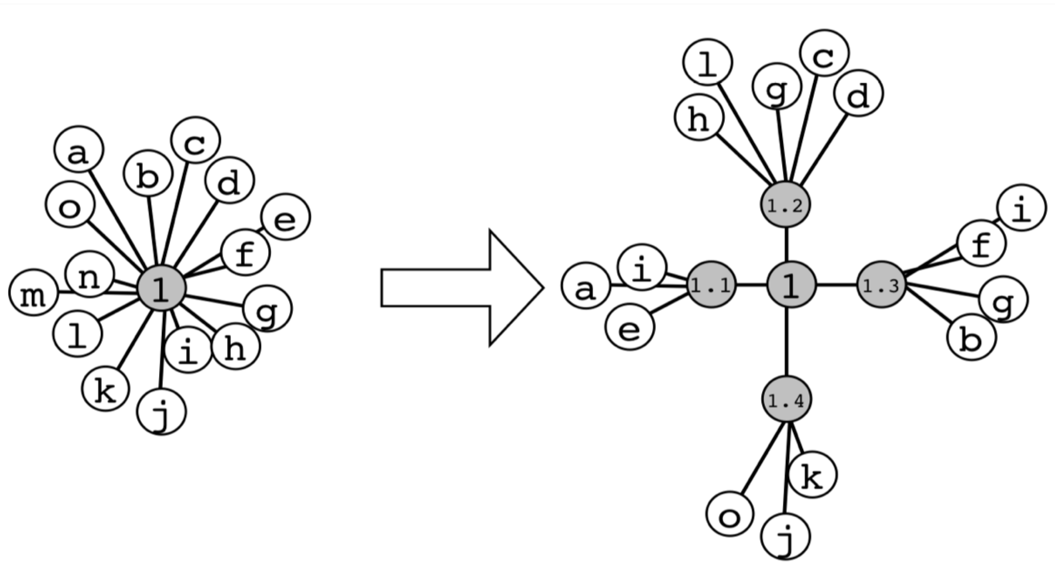
```python
''''' Pratik Agarwal  \
* Aman Bhayana '''
from pyspark import SparkConf, SparkContext
import sys

def lsplit(edge):
    line = edge.split(" ")
    v = [int(x) for x in line]
    yield (v[0], v[1])
    yield (v[1], v[0])

def largestarmap(data):
    u = data[0]
    v = data[1]
    yield (u, v)
    yield (v, u)

def smallstarmap(data):
    u = data[0]
    v = data[1]
    if u >= v:
        yield (u, v)
    else:
        yield (v, u)

def largestarreduce(data):
    u, v = data
    t = list(v)
    t.insert(len(t), u)
    m = min(t)
    for x in t:
        if x > u:
            yield x, m

def smallstarreduce(data):
    u, v = data
    t = list(v)
    t.insert(len(t), u)
    m = min(t)
    for x in t:
        if x != m:
            yield x, m

if __name__ == '__main__':
    conf = SparkConf().setAppName("RDDcreate")
    sc = SparkContext(conf=conf)
    rddSS = sc.textFile(sys.argv[1]).flatMap(lsplit).groupByKey().flatMap(largestarreduce).distinct()
    convergence = 1
    while convergence != 0:
        tempc = 1
        while tempc != 0:
            rddLS = rddSS.flatMap(largestarmap).groupByKey().flatMap(largestarreduce).distinct()
            rddSS = rddLS.flatMap(largestarmap).groupByKey().flatMap(largestarreduce).distinct()
            tempc = (rddLS.subtract(rddSS).union(rddSS.subtract(rddLS))).count()
        rddSS = rddLS.flatMap(smallstarmap).groupByKey().flatMap(smallstarreduce).distinct()
        convergence = (rddLS.subtract(rddSS).union(rddSS.subtract(rddLS))).count()
    print(convergence)
    op = rddSS.collect()
    print(op)
```

**The above code was actually tested on UB CCR to simulate a distributed environment functionality in Apache SPARK.*

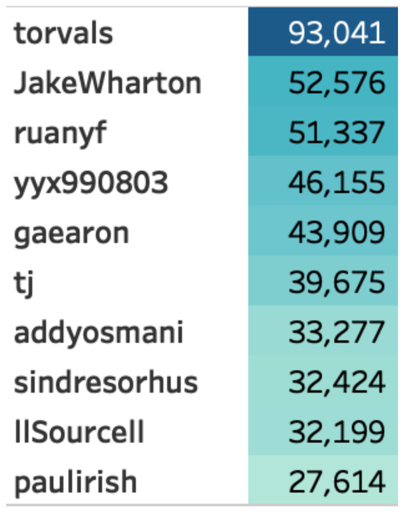
Expected Outcome:

This will print the node which has most followers and thus our problem statement can be achieved using the RDDs provide by Apache SPARK. The below illustration presents the output format:

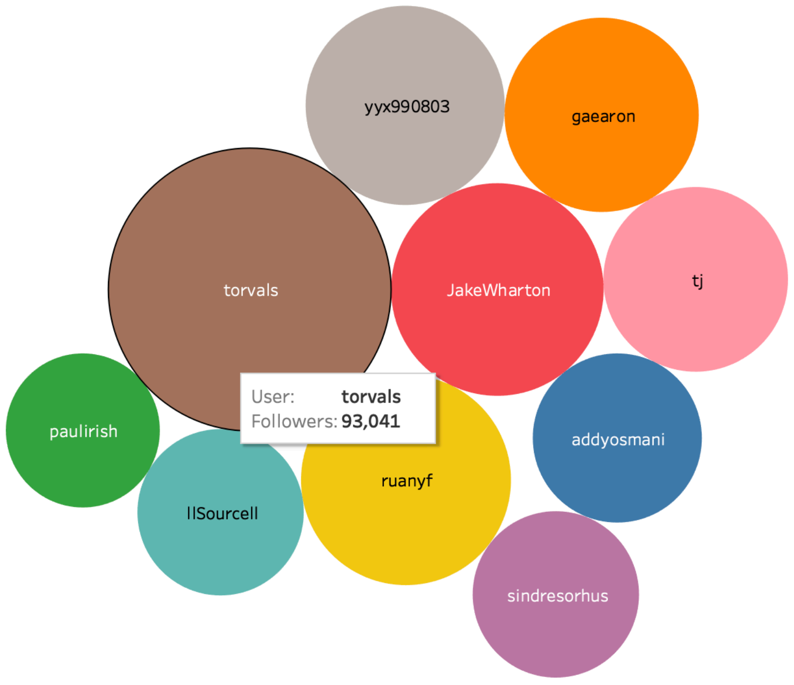
Node 1 represents the user with most followers:



Some other sample visualizations:



torvals	93,041
JakeWharton	52,576
ruanyf	51,337
yyx990803	46,155
gaearon	43,909
tj	39,675
addyosmani	33,277
sindresorhus	32,424
llSourcell	32,199
paulirish	27,614



Summary and Future Extensions

From all the illustrations presented above, it is evident that Apache SPARK provides a powerful mechanism for getting the user with most followers on GitHub. This algorithm with slight modifications can be used on any Social Networking website to determine the person or user with most influence. There can be several extensions to our project. This algorithm can also be used to determine the most used programming language on GitHub, top open source projects, fastest growing project, open source contributions made by employees of different organizations etc. The list can go on and one. The takeaway from this project is that mechanisms of Lazy computation provided by the RDDs in Apache SPARK provides a powerful tool using which datasets of large scale can be easily processed and solved.

References

[1] R. Kiveris, S. Lattanzi, V. Mirrokni, V. Rastogi, and S. Vassilvitskii, “Connected Components in MapReduce and Beyond,” in Proc. ACM Symposium on Cloud Computing (SOCC), 2014, pp. 1-13

[2] H. Karau, A. Konwinski, P. Wendell, M. Zaharia, “Learning Spark: Lightning-Fast Big Data Analysis,” O’Reilly Media, 2015, ISBN-13: 9781449358624.