

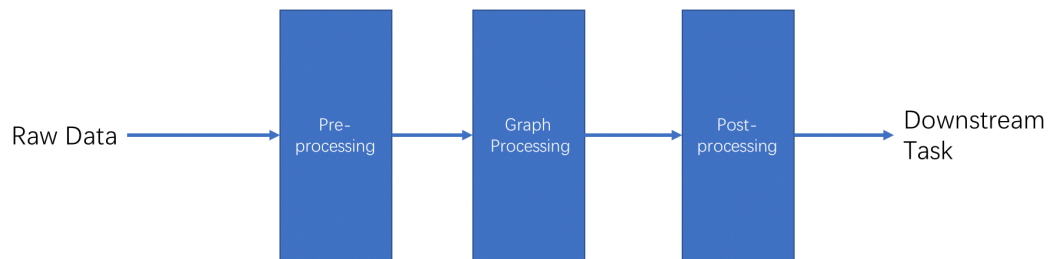
Final Project

Due: June 18, 11:59pm

The task is not difficult. Start earlier, or maybe you would need to wait for computing resources of servers as the deadline approaches

[Description]

In this assignment, you need to implement three parts of the typical pipeline of graph processing in real world scenario:



(1) Pre-processing:

In real word scenario, the raw graph data usually contains some sensitive information (like username of accounts, url, etc). So before used in the next step, these data need to be pre-processed to convert the raw identifier of vertices to the integer ID which contains no sensitive information.

[What you need do]

In brief, you need to implement a *mapping* program: raw identifier \rightarrow integer id (from $0 \sim |N|-1$ or $1 \sim |N|$, $|N|$ is the number of vertices).

The raw identifiers of the graph data we provided are integer but not in the range $[0, |N|-1]$. So, you can treat it as integer type or string type when you implement this program.

(2) Graph processing:

In this part, what you need to do is similar to some assignments before. You need implement two graph processing algorithms on the real-world dataset.

[What you need do]

1. Implement the pagerank algorithm with any language or framework you like and test for performance.

PageRank Iteration:(<https://en.wikipedia.org/wiki/PageRank>)

Iterative [edit]

At $t = 0$, an initial probability distribution is assumed, usually

$$PR(p_i; 0) = \frac{1}{N}.$$

where N is the total number of pages, and $p_i; 0$ is page i at time 0.

At each time step, the computation, as detailed above, yields

$$PR(p_i; t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)}$$

where d is the damping factor,

or in matrix notation

$$\mathbf{R}(t+1) = d\mathcal{M}\mathbf{R}(t) + \frac{1-d}{N}\mathbf{1},$$

where $\mathbf{R}_i(t) = PR(p_i; t)$ and $\mathbf{1}$ is the column vector of length N containing only ones.

The matrix \mathcal{M} is defined as

$$\mathcal{M}_{ij} = \begin{cases} 1/L(p_j), & \text{if } j \text{ links to } i \\ 0, & \text{otherwise} \end{cases}$$

i.e.,

$$\mathcal{M} := (K^{-1}A)^T,$$

where A denotes the [adjacency matrix](#) of the graph and K is the diagonal matrix with the outdegrees in the diagonal.

The probability calculation is made for each page at a time point, then repeated for the next time point. The computation ends when for some small ϵ

$$|\mathbf{R}(t+1) - \mathbf{R}(t)| < \epsilon,$$

i.e., when convergence is assumed.

Run the aforementioned iterative process for 20 iterations (**assuming $d = 0.8$**).

Compute the top 5 node IDs with the highest PageRank scores.

2. Implement the TrustRank algorithm and test.

TrustRank is very similar to PageRank, which calculates the trust value (TR) of any vertex according to the given white list. The higher the trust value, the less likely the vertex is an illegal vertex. The main difference is at the initial phase. In PageRank, the initial PR value of all vertices are set to $\frac{1}{N}$, where N is the number of all vertices. But in TrustRank, the initial TR value of the vertices in white list are set to $\frac{1}{\#\{\text{nums of vertrices in white list}\}}$, and the initial value of other vertices are set to 0. The iteration process later is same as which

in Pagerank (assuming $d = 0.8$).

You need compute the TR value of all nodes after 20 iterations and save in the file. You need randomly select 100 benign nodes as white list before you test this algorithm and save this white list as file.

(3) Post-processing:

After the 2nd part, you may get some results like top 5 node IDs with the highest PageRank scores, or all node IDs with TrustRank scores. **In this part, you need to do the ‘re-mapping’ operation: convert node ID (after pre-processing) in the results to raw identifier.**

You can use C/C++, Java, Scala or Python as you want.

You can also use frameworks learned before (like MPI, MapReduce, Spark, GridGraph, Gemini) as you want.

Of course, you can use different languages or frameworks in each part of this assignment.

[Notice]

It's better that backup your code in time (like using git) in case the cluster crashes.

[Datasets]

Here is some basic information of datasets used in this assignment.

The graph data has been put on the server under the path **/nfs/share/data**.

```
vagrant@bd1:/nfs/share/data$ ls
com-lj.ungraph.txt  wiki-Vote.txt
```

Name	Type	Nodes	Edges
wiki-Vote.txt	Directed	7,115	103,689
com-lj.ungraph.txt	Undirected	3,997,962	34,681,189

If you want test on your own machine first, you can download datasets from:

<https://cloud.tsinghua.edu.cn/d/2cb95ce6af2f4a34b222/>

[Graph Format]

The raw graph data are edge lists of unweighted graphs in text form. If the graph is undirected, each unordered pair of nodes appear only once.

Each line of the file represents an edge is:

<source_identifier destination_identifier > (separator is one space)

For example:

```
> head com-lj.ungraph.txt
0 1
0 2
0 31
0 73
0 80
0 113619
0 2468556
0 2823829
0 2823833
0 2846857
```

[Output Format]

(1) pre-processing

After this part, you should get the graph data after pre-processing and the related mapping file.

The graph data can be stored in text form or binary form which depends on your implementation of graph algorithm later. For undirected graph, maybe you need store each edge twice where consider an edge as two directed edges depends on your implementation of graph algorithm and framework.

The related mapping files can be any format as you like, but remember give detailed description of your format. Here is an example format:

<raw_identifier new_identifier > (separator is one space)

(2) graph processing

After this part, you should get some results like top 5 node IDs with the highest PageRank scores, or all node IDs with TrustRank scores. Save these results in the related files.

(3) post-processing

For this part, you should get the files in the format same with which in the 2nd part, but convert the node ID (after pre-processing) in the results to the related raw identifier.

The files you need submit are:

1. all the result files of wiki-vote dataset (with the whitelist you generated).
2. the PageRank result file of com-lj.ungraph.txt.

[Environment]

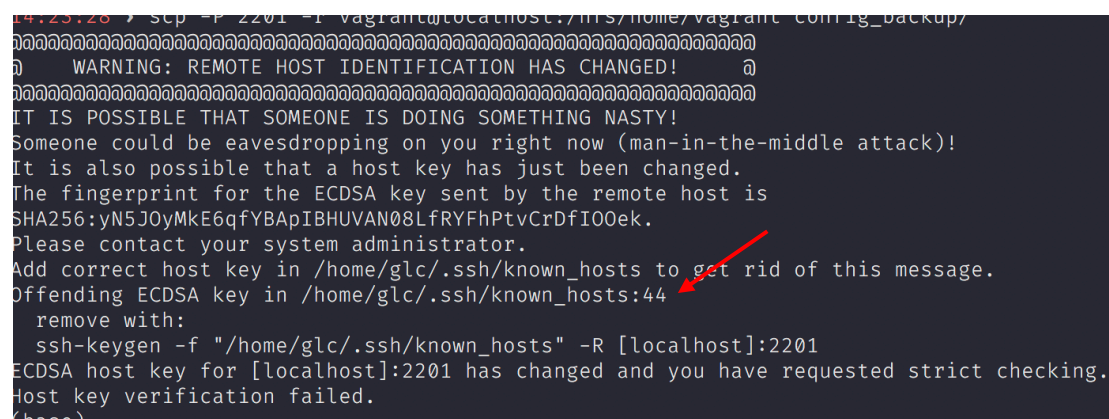
You will need to run code on the cluster machines for this assignment.

Server IP Address: **166.111.68.163**

Port: **2201**

[NOTICE]

The port is same with which in hw3 (MPI), but actually this cluster is different from which in hw3. So, when you first try login the server via ssh, you may get some error information like below:

A terminal window showing an SSH connection attempt. The prompt is '14.23.28 ~' and the command is 'scp -P 2201 -r vagrant@localhost:/nfs/home/vagrant_config_backup/'. The output shows a warning: 'WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!'. It explains that it's possible someone is doing something nasty or a host key has changed. It provides the fingerprint for the ECDSA key: 'SHA256:yN5JOyMkE6qfYBApIBHUVAN08LfRYFhPtVcrDfIO0ek.' and instructs to contact the system administrator. It also says to add the correct host key to the known_hosts file to get rid of the message. The offending ECDSA key is listed as 'Offending ECDSA key in /home/glc/.ssh/known_hosts:44'. A red arrow points to the number '44'. Below this, it says 'remove with:' followed by the command 'ssh-keygen -f "/home/glc/.ssh/known_hosts" -R [localhost]:2201'. The final output is 'ECDSA host key for [localhost]:2201 has changed and you have requested strict checking. Host key verification failed.' and the prompt returns to '(base) ~'.

If you get this error, you can just edit the *known_hosts file* under *~/.ssh* on your own machine and delete the related line. Like the figure above, you should delete the 44th line of the *known_hosts* file.

The cluster consists of 4 machines you can use for this assignment.

The specification of each machine is:

Operating System	Ubuntu Server 16.04.1 LTS 64-bit
CPU	Intel(R) Xeon(R) CPU E5-2680 8 cores 1 thread per core
Memory	32 GB
C/C++ Compiler	gcc 7.5
MPI version	MPICH 3.3.2

You can use `ssh {username}@166.111.68.163 -p 2201` to login the server.

Your username is {your_student_id} and original password is {your_student_id}.

You must login the server with the original password at your first login.

The hostnames of 4 machines are: (maybe used when you want to test your own MPI program under different configurations):

bd1 (the head node you log in) bd2 bd3 bd4

Your HOME directory is shared among the 4 machines.

The specification of some software installed on the cluster:

Python 3.7.7

Hadoop 3.1.3 (running on 4 nodes in the cluster mode)

Spark 2.4.5 (cluster mode)

Scala 2.11.12

OpenJDK 1.8

Maven 3.6.3

The repo of GridGraph and Gemini have been put on the path “/nfs/share/”. If you need use these, you can first copy the repo to your own directory.

[NOTICE]

If you need use hdfs/mapreduce/spark, here are some additional information you should pay attention to:

The master of Hadoop or Spark is **bd2**, not bd1. So the HDFS path in your code should begin with “**hdfs://bd2:9000/**”.

[For spark user]

And if you want run your spark job in the cluster mode, you need to add parameter called ‘**—master spark://bd2:7077**’ when using spark-submit. Here is a simple example:

```
spark-submit --name test --master spark://bd2:7077 --executor-memory 2G --  
total-executor-cores 2 SimpleApp.py
```

You can change “spark://bd2:7077” to “local[*]” and just run your spark-job in the local server.

If you use JAVA/Scala to write Spark program, you should use “Maven” to manage and compile the Java/Scala project. You can refer to the related section in hw10 and the code skeleton offered in hw10 to organize your program.

[Grades]

70 points: Submit on time and code of each part can be executed successfully.

20 points: Detailed report.

10 points: For performance. This part depends on your performance of each part.

[Hand-in]

Please submit your assignment containing your report, code and the **result file (see section [Output Format])**.

Please describe your solution in detail in your report. Besides, please tell me how to compile and run your program successfully.

If you have any questions about this assignment, please contact me:

Email: lincheng96@qq.com