# **LSGi Demo: Deployment, Configuration and Testing Document**

Tere Gonzalez, Janneth Rivera, Hernan Laffite, Fei Chen, Krishna Viswanathan  
Dec 17. 2015, Updated July 21, 2016

Contents

[**LSGi Demo: Deployment, Configuration and Testing Document** 1](#_Toc456890891)

[**1.** **Overview** 1](#_Toc456890892)

[**2.** **LSGi Package Content** 2](#_Toc456890893)

[**3.** **LSGi Deployment and Configuration** 3](#_Toc456890894)

[**3.1.** **Initial Requirements** 4](#_Toc456890895)

[**3.2.** **Download the package** 4](#_Toc456890896)

[**3.3.** **Update Hosts File** 5](#_Toc456890897)

[**3.4.** **SSH Passwordless** 5](#_Toc456890898)

[**3.5.** **Deploy package** 6](#_Toc456890899)

[**3.6.** **Configure the package** 8](#_Toc456890901)

[**3.7.** **Run test cases** 8](#_Toc456890902)

[**4.** **Manual Steps to Run single node demo and verify the results** 10](#_Toc456890903)

[**4.1.** **How to verify that single node run was successfully** 11](#_Toc456890904)

[**5.** **Run multi-node demo** 12](#_Toc456890905)

[**6.** **Query inference states** 13](#_Toc456890906)

[**7.** **History** 16](#_Toc456890907)

## **Overview**

This document summarizes how to configure LSGi engine and run the demo in order to test graph inference models running in l4tm. The document describes methods to deploy, configure and test the engine.

**About LSGi**

The LSGi engine uses a graphical inference algorithm to compute a joint probability of the graph given a small evidence in order to predict unknown events. Each vertex is associated to a random variable which denotes a discrete event or state. Our approximate inference algorithm infer a new state using iterative computation over all the vertices of the graph until convergence is reached.

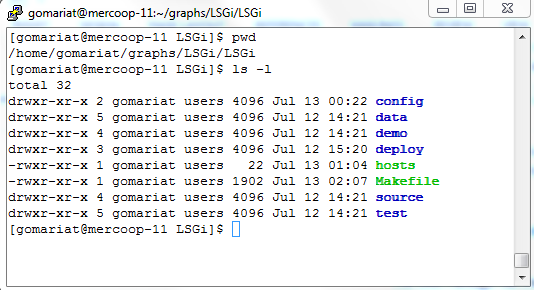
**About the Architecture of LSGi**

The system is designed to run in a multicore and multinode environment using a graph-data parallel approach. The graph is partitioned over N multi-nodes and the engine launch N processes to compute the inference for each partitioned graph –local graph. In order to compute the global inference, the processes push their local predicted states to other processes; at the same time the processes pull from the other remote states to their local computation. The LSGi engine is designed to use a shared memory approach as communication medium to minimize communication overheads. The states can be stored in: a) **/dev/shm** while running multiple processes in a single server; or b) in the librarian file system **/lfs/** – a high bandwidth and low latency network file system while running the processes in multiple computing nodes connected.

For further background on conceptual design and model considerations, please see the documents posted in our repository.

## **LSGi Package Content**

The LSGI Package has the following modules:



* **/config.** Scripts to configure remote nodes: compile and ssh passwordless.
* /**data**: It contains input graphs as binary files, configuration files and will host local outputs of the inferences like local states and statistics.
* /**demo**. It contains scripts to demo the existing functionality for the project that comprises:

-run a single node inference  
-run multimode inference  
-run query client to request inference states

* **/deploy**. Scripts to deploy the local checkout to remote nodes.
* /**docs**: It contains multiple documents that describe the inference engine.
* /**source**: It contains the main source code for the modules: inference, query service and query client. To compile and generate the executables it should be executed “make clean” and then “make” commands.
* **/test.** It contains several basic test cases to test librarian access, pmem persistence and sync that are fundamental to verify that the shared states are working in the environment.
* **hosts:** main file to configure the computing nodes.
* **Makefile:** main configuration and deployment goals

## **LSGi Deployment and Configuration**

LSGi package includes the steps to deploy and configure the system. It assumes a set of computing nodes which will have a replica of the full package. All computing nodes should have access to a shared memory like /lfs or /dev/shm/. In case of the package is not deployed in the computing nodes, the Makefile includes a goal for deploy which copies to the list of nodes included on the hosts file. One of the nodes, the first of the host file configuration, will be named as Master node and the configuration step will create automatically an ssh passwordless between the master and the rest of the nodes. Finally, once it is configured, LSGi package includes 2 main test cases to verify if the configuration has been done successfully.



LSGi Configuration & Deployment Diagram

Once the computing nodes are configured, the master node is responsible for launching the inference jobs over the computing nodes. Each node will execute a local inference computation and access data from librarian to synchronize global inference states. The master node is also responsible for launching a query service which is a TCP/IP service that also access librarian data to retrieve results to the LSGi web service in an external tomcat server.



Operational LSGi Deployment Diagram

### **Initial Requirements**

1. Git: The LSGi package is on GitHub HPE.

$ sudo apt-get install git

1. g++ compiler >= 4.8
2. Lipmem
3. Pthreads
4. **Valid user to login to from host to computing nodes, and from master to other nodes.**

### **Download the package**

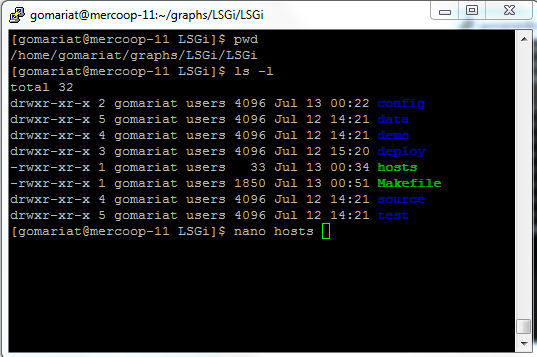
First, you need to checkout the LSGi package to have access to the deployment and configuration scripts. The package is located at the following URL:   
https://github.com/HewlettPackard/LSGI/

#Checkout the package from GitHub  
  
$ git clone https://github.com/HewlettPackard/LSGI/  
$ cd LSGi/

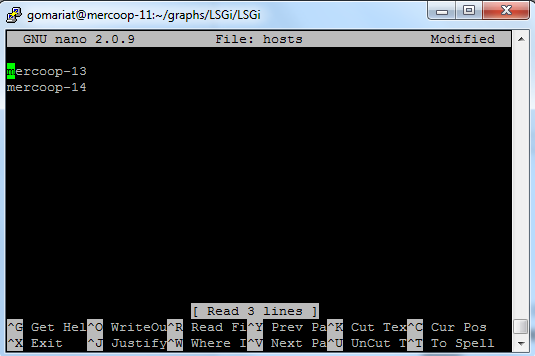
### **Update Hosts File**

The hosts file should contain the computing nodes that are going to be configured in the system. In the previous checkout folder, go to LSGi/ directory which is the **LSGi root**.

$ cd LSGi/  
$ nano hosts



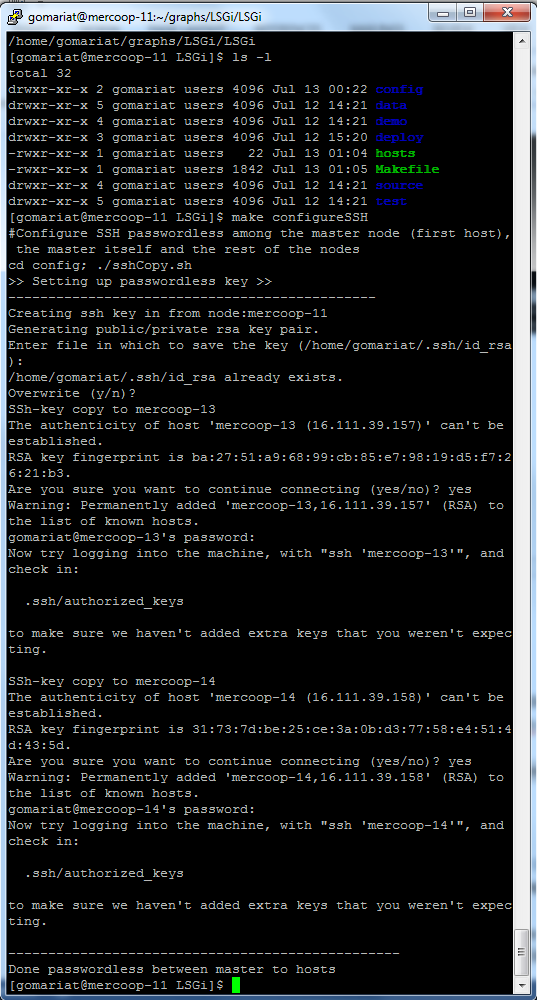
Add the host name or IP of the computing nodes as list, one in each line as follows:



### **SSH Passwordless**

In order to proceed with the deployment and configuration, it is required to setup passwordless SSH from host server to configuration nodes. To generate ssh key and copy to the computing nodes, go to LSGi root folder and run the script as follows:

$ make configureSSH

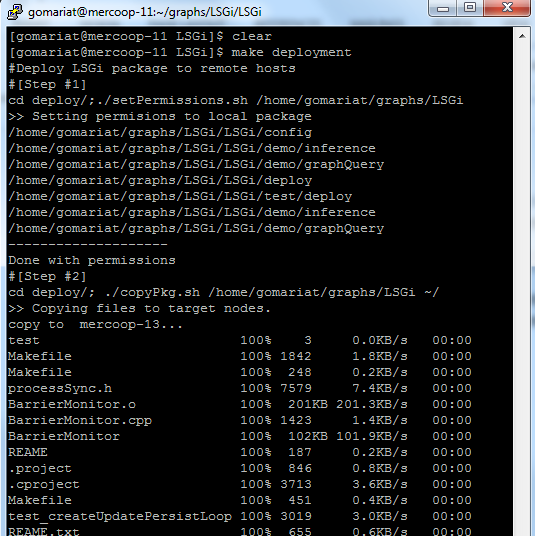


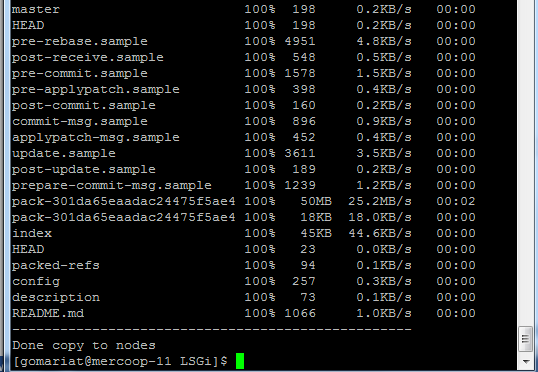
### **Deploy package**

If the computing nodes do not have the LSGi package, you can deploy it using the following command:

$ make deployment

This deployment goal use the current path as source folder to copy. The destination folder is the home directory of the user. If you would like to change the paths, you can edit the Makefile to change the **LSGi\_PATH\_SRC** and **LSGi\_PATH\_DEST**.



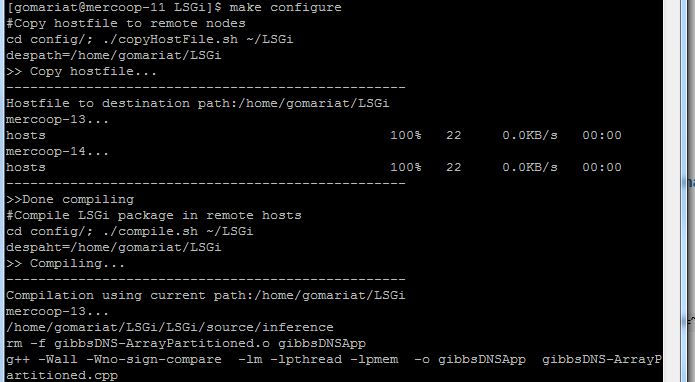


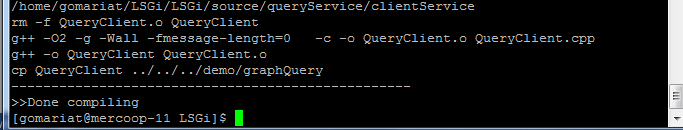
### **Configure the package**

Once the package exists on the remote nodes, you can run the configure script as follow goal:

$ make configure

This configure goal will copy host file changes into the computing nodes and compile remote LSGi packages to have it ready for test cases and demo.





After the goal, the compilation should be done with 0 Errors. Otherwise, check the configuration path.

### **Run test cases**

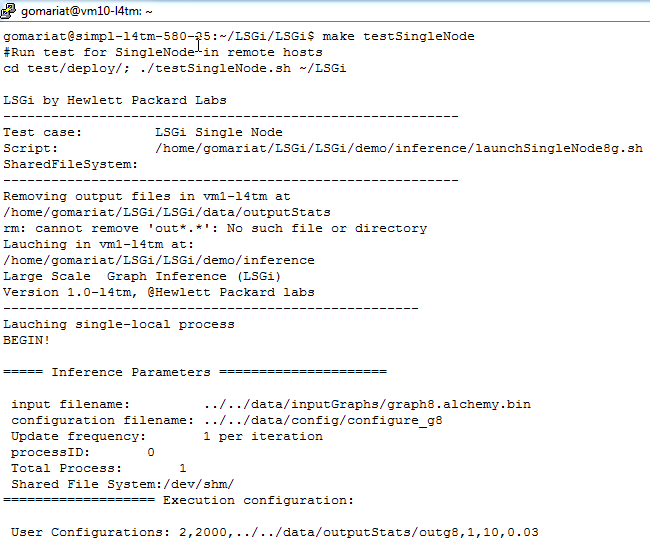
#### Fists, add passwordless between master node and the other compute nodes

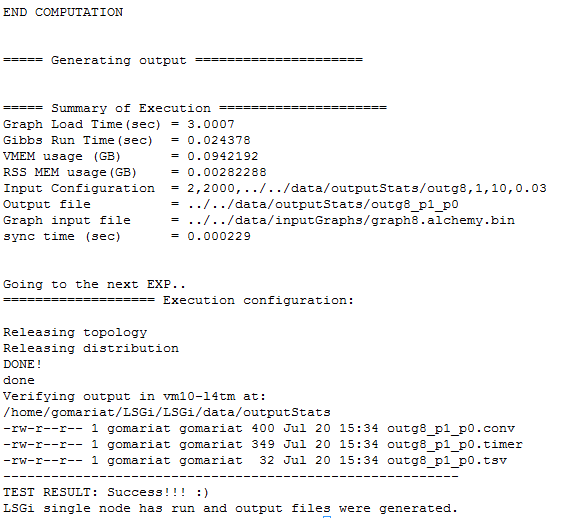
$ ssh <masterNode> #the first node of the hosts file  
$ cd ~/LSGi/LSGi  
$ make configureSSH

#### Run Single Node Test

$ make testSingleNode

The single node launcher will execute the inference in verbose mode. The program outputs the trace on the console. If everything is correct, the user should see the legend “***TEST RESULT: Success!!!***” on the summary of the execution as follows:

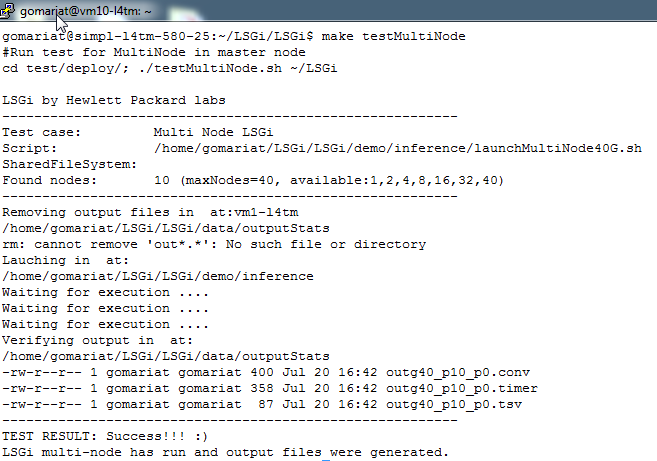




#### Run Multi Node test

$ make testMultiNode

The multi node launcher will execute the inference in verbose mode. The program outputs the trace on the console. If everything is correct, the user should see the legend “***TEST RESULT: Success!!!***” on summary of the execution as follows:

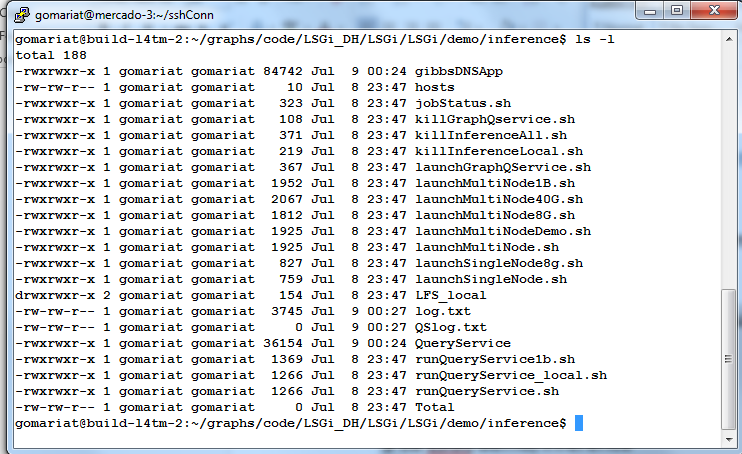


## **Manual Steps to** **Run single node demo and verify the results**

First, clean up previous run results and then you can run manually the launcher of 8-vertex graph as follows:

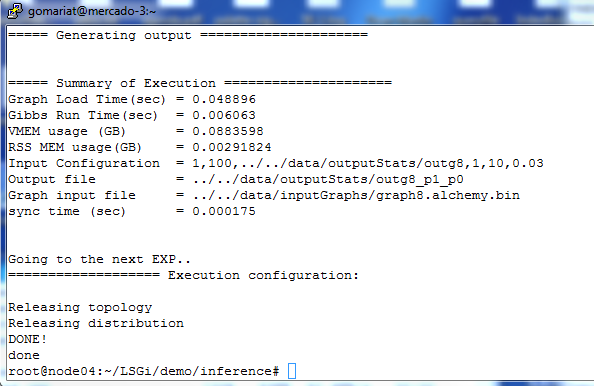
# Clean and remove previous run results  
$ rm LSGi/data/outputStats/ \*  
  
#run small graph – 8 vertex graph  
$ cd LSGi/demo/inference  
$ ./launchSingleNode8G.sh

You can browse the content of the demo folder, which include all the scripts to launch, monitor, and kill the inference jobs.



### **How to verify that single node run was successfully**

The single node launcher will execute the inference on the local machine in verbose mode. The program outputs the trace on the console. If everything is correct, the user should see the summary of the execution as follows:



1. Verify the output results:

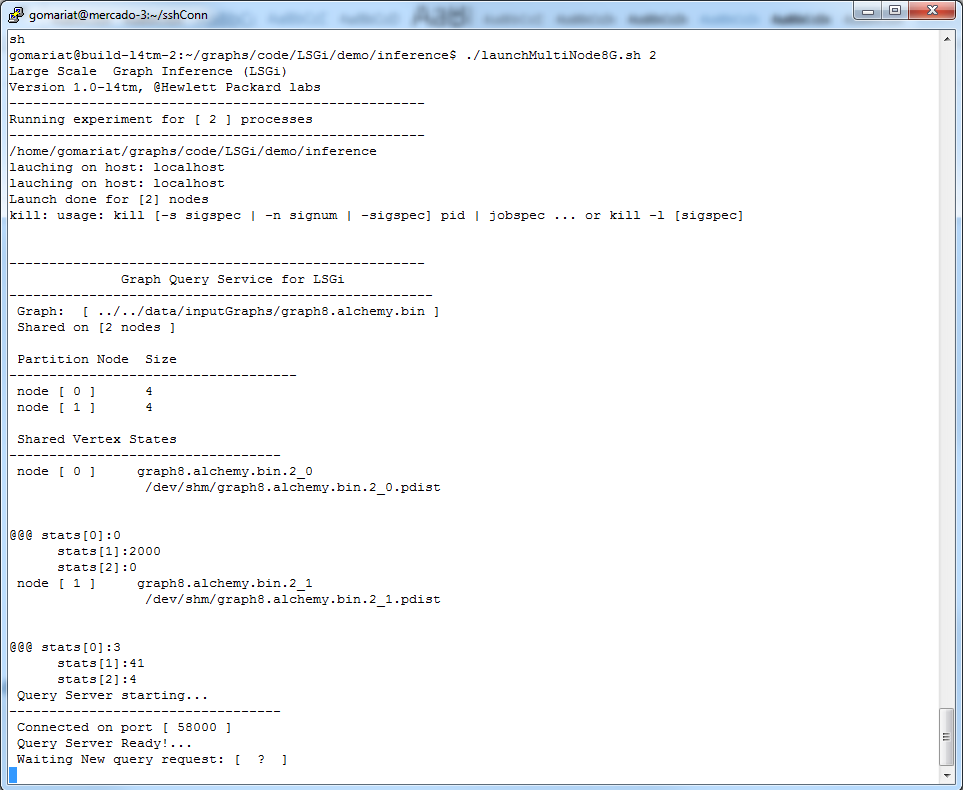
#go to the output folder   
$ ls ../../data/outputStats/outg8\_\*  
$ cat ../../data/outputStats/outg8\_p1\_p0.tsv

0  
843  
1336  
517  
346  
2000  
712  
862

#the columns is yellow should be 0 and 2000, the other values should vary but they should be very close.

## **Run multi-node demo**

$ cd LSGi/demo/inference  
  
#specify the number of process to run,   
#for example 4 or 2 , one in each node of the host file  
$ ./launchMultiNode8G.sh 2



The user should see the trace of the process launched as is showed above. This script will also startup a Query Service that waits for user queries and retrieve the graph states requested. See next section.

## **Query inference states**

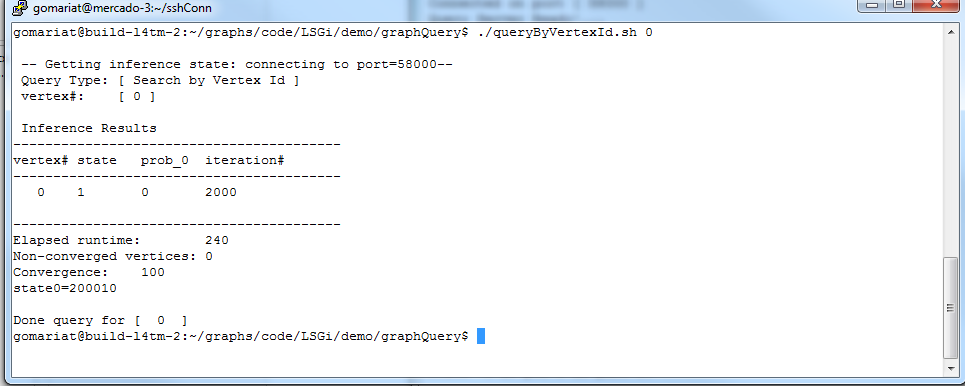
Once the inference is running, the user can request states of the vertices as follows:

1. **Query By Vertex identifier**

Test 1:

$ cd LSGi/demo/graphquery/

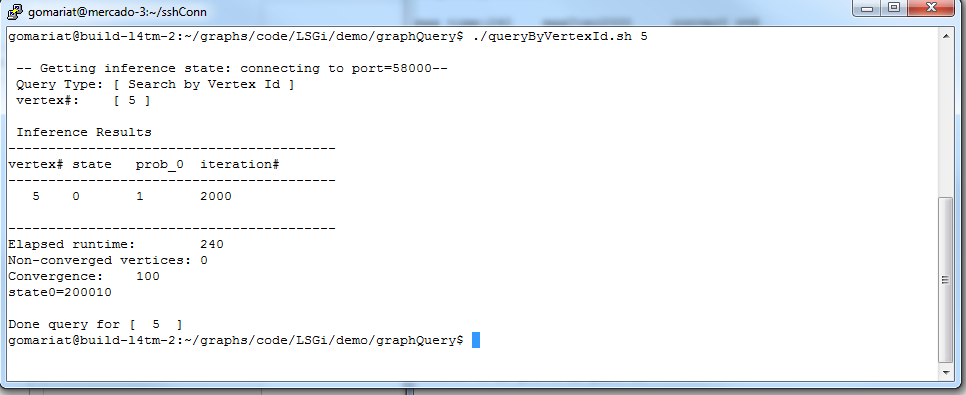
#query state for specific vertex id  
$ ./queryByVertexId.sh 0



The iteration number should go up to 2000 when the inference finished and the state is =1 and prob\_0=0 for vextex 1. If you query before the inference has finished, if everything is running, the iteration should be changing up to 2000.

Test 2.

#query state for specific vertex id  
$ ./queryByVertexId.sh 5



The iteration number should go up to 2000 when the inference finished and the state is =0 and prob\_0=1 for vextex 5. If you query before the inference has finished, if everything is running, the iteration should be changing up to 2000.

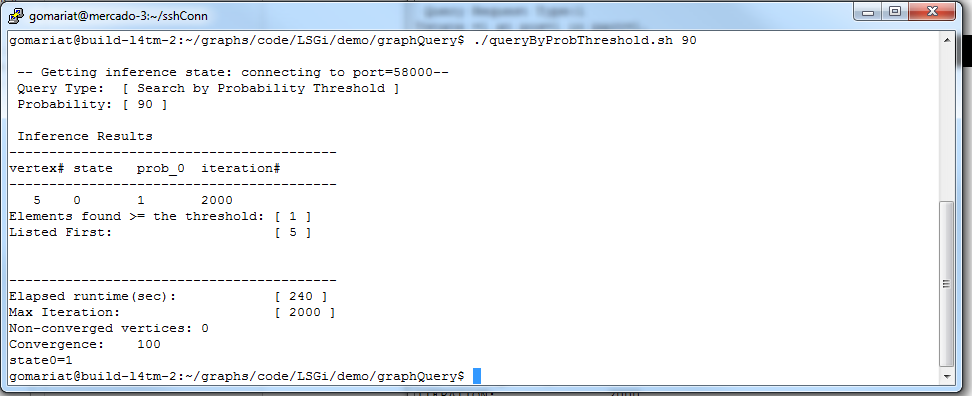
1. **Query By Probability Threshold.**

Test 1:

$ cd LSGi/demo/graphquery/

#query state for a subset of vertices which probability on infected above the threshold  
#query for 90 probability of state 0  
$ ./queryByProbThreshold.sh 90

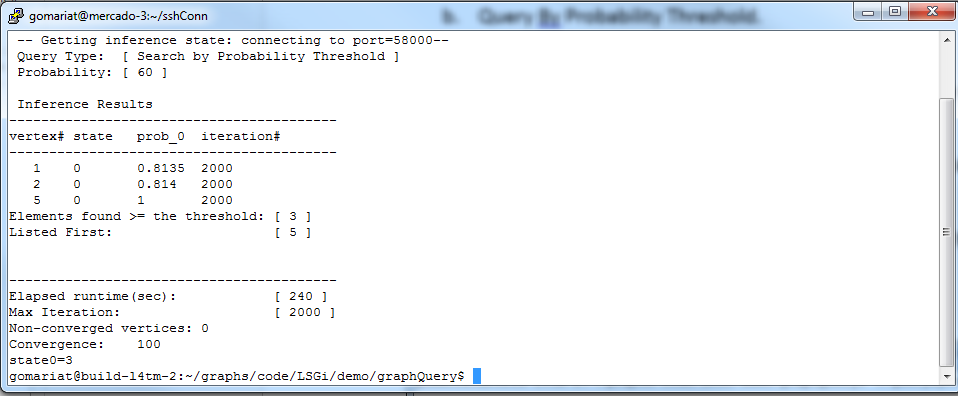
The output should display only 1 element over 90% probability of being state=0;



Test 2:

$ ./queryByProbThreshold.sh 60

The output should display 3 elements over 60% probability of being state=0.



## **History**

Created Dec 14, 2015 Version 1.0  
Updated June 2, 2016 version 1.1  
Updated July 8, 2016 version 1.2

Updated July 12, 2016 version 2.0

+update deployment method.

+ Change gitrepo URL  
+ Add results verification

Updated July 15, 2016 version 2.1

+Correct typos

+better screenshots