1.Background

A gossip protocol is a procedure or process of computer-computer communication that is based on the way social networks disseminate information or how epidemics spread.

2.Project goal

1.Implement gossip protocol over nine nodes
2.Simulate gossip protocol over more than 1,000 nodes on a machine.

3.Implemente gossip protocol on a small-scale cluster

We first implement gossip protocol over nine nodes. In this section, we describe how to implement it and how it works.

1)Method

RandomInteger object, which is responsible for generate random integer between a determined range.

send/receive object, the former sends UDP message to other nodes while the later receives UDP message from other nodes.

IfPortUsed object, which assigns UDP ports and make sure that each port is unique.

Node object, which is regarded as a node and stores the version of message. It chooses a neighbor randomly to send its message per 0.5~1.5 sec by calling *send* and receive its neighbor's message by calling *receive.* Then it compare the version of message it stores with the one it received from its neighbor just now to update to version, which means the node replace the message if the version of it is older than the one from other node.

Seed object, which executes every 8 seconds to select a node randomly and increment the version number of its message.

PrintInfo object, which prints the version of all the nodes per-sec.

2)Implementation

We generate nine ports that are different from each other and then bind them to nine sockets which are assigned to nine nodes. Each node runs on a single thread which create two threads one sends UDP message to its neighbor and another is responsible for receive message and update its version to the higher one. Create a thread to run a *seed* object which select a node randomly per 8 seconds and update its message version.

3)Analysis

Figure1 shows the topology of the cluster, which has nine nodes and eleven edges. Open the command window and run the executable jar file, Figure2 shows the results, as you can see, it updates the message version of one of the nodes and it spreads like

epidemics to other nodes in the cluster.

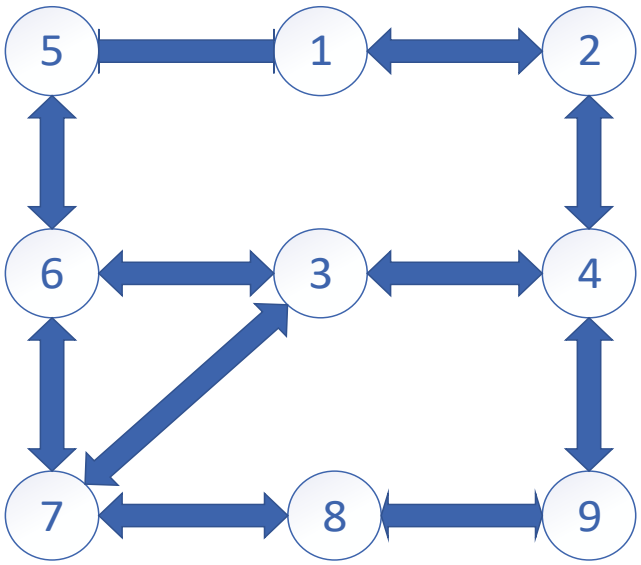

Figure1: The topology of cluster

```
Node 8's version is update to 2.
Version of Node 1: 1
Version of Node 2: 1
Version of Node 3: 1
Version of Node 4: 1
Version of Node 5: 1
Version of Node 6: 1
Version of Node 7: 1
Version of Node 8: 2
Version of Node 9: 1

Version of Node 1: 1
Version of Node 2: 1
Version of Node 3: 1
Version of Node 4: 1
Version of Node 5: 1
Version of Node 6: 2
Version of Node 7: 2
Version of Node 8: 2
Version of Node 9: 1

Version of Node 1: 1
Version of Node 2: 1
Version of Node 3: 1
Version of Node 4: 2
Version of Node 5: 1
Version of Node 6: 2
Version of Node 7: 2
Version of Node 8: 2
Version of Node 9: 2
```

```
Version of Node 1: 2
Version of Node 2: 2
Version of Node 3: 2
Version of Node 4: 2
Version of Node 5: 2
Version of Node 6: 2
Version of Node 7: 2
Version of Node 8: 2
Version of Node 9: 2

Node 6's version is update to 3.
Version of Node 1: 2
Version of Node 2: 2
Version of Node 3: 2
Version of Node 4: 2
Version of Node 5: 2
Version of Node 6: 3
Version of Node 7: 2
Version of Node 8: 2
Version of Node 9: 2

Version of Node 1: 2
Version of Node 2: 2
Version of Node 3: 2
Version of Node 4: 2
Version of Node 5: 3
Version of Node 6: 3
Version of Node 7: 2
Version of Node 8: 2
Version of Node 9: 2
```

Figure2. the result of implementation

4.Simulate a large-scale cluster

The small-scale cluster we described above is a real demo to implement gossip protocol on different nodes which communicate by UDP. However, it is almost

impossible to create thousands of threads which communicate by UDP on a single machine, so we use a special way to simulate a large-scale cluster to implement gossip protocol. And we show the spread process on a real-time graphical interface.

4.1 method

1)Creating a class named *Host* and every host is an object of it which has attributes as follow:

Id distinguish one host from others

Friends contains its neighbors in the cluster

Crashed represents its state

Version is the highest version of the message it stores

2)Conspicuously the class Host is far too enough to implement the gossip protocol, as it cannot send message. Given that creating thousands of threads which transmit message by UDP in one machine is hard to achieve, we need another class named *God* to help the nodes communicate with each other just like a bus. God also has the ability to add new message to the cluster that is updating the version of message, and select nodes randomly to make them crashed or repaired. we present some of its attributes as follow:

hosts stores all the hosts (or nodes) in a list.

scale represents the scale of the cluster and we can add or delete nodes to change the scale of the nodes by change the value of this parameter. Its initial value is 1000 which means there are 1,000 nodes in the cluster.

MessageBox is a Blocking Queue to store message in all the nodes, each element in it is the highest version of message of the corresponding node , and we will describe how it helps in detail later.

Sparse is the sparse degree of the topology and we can set up how many neighbors the node has through this parameter.

UdpLossRate means the packet loss rate because we aim to simulate the nodes communicate with each other by UDP which, in fact, is impossible to achieve in our computer and UDP is unreliable.

Message creator is responsible to collect the messages from the hosts.

Message handler helps the hosts hand message.

Actually, as we have said, that the nodes are not able to communicate with each other directly, however they can achieve it through God. In real mode, every round each node sends message to one of its neighbor at the same time. Every round *God* get message from messageBox sequentially, for example God first get the message in position 0 whose corresponding node is node 0, and God select one of the neighbors of node 0 randomly to send the message, and at that time god works as the agent of node 0. By this way we success to simulate all nodes in the cluster send message to one of their neighbors simultaneously in every round which is achieved through *God*

sequentially.

As we have mentioned above, we also simulate UDP packet loss rate by set up the parameter *UdpLossRate*. Each round *God* works as the agent of every nodes in the cluster to send message, however, in real environment message could be lost sending by UDP. So, we select *UdpLossRate*scale* nodes each round that *God* does not send message when works as their agent, which can be regarded as packet loss.

4.2 Instruction for use

1)Figure3 is the graphical interface of our project, x coordinate represents the number of round and y coordinate is the number of host affected by the highest version message. The line chart shows dynamically as the rounds increase, the number of hosts have lower version decrease while that with higher version increase till all the hosts are affected.

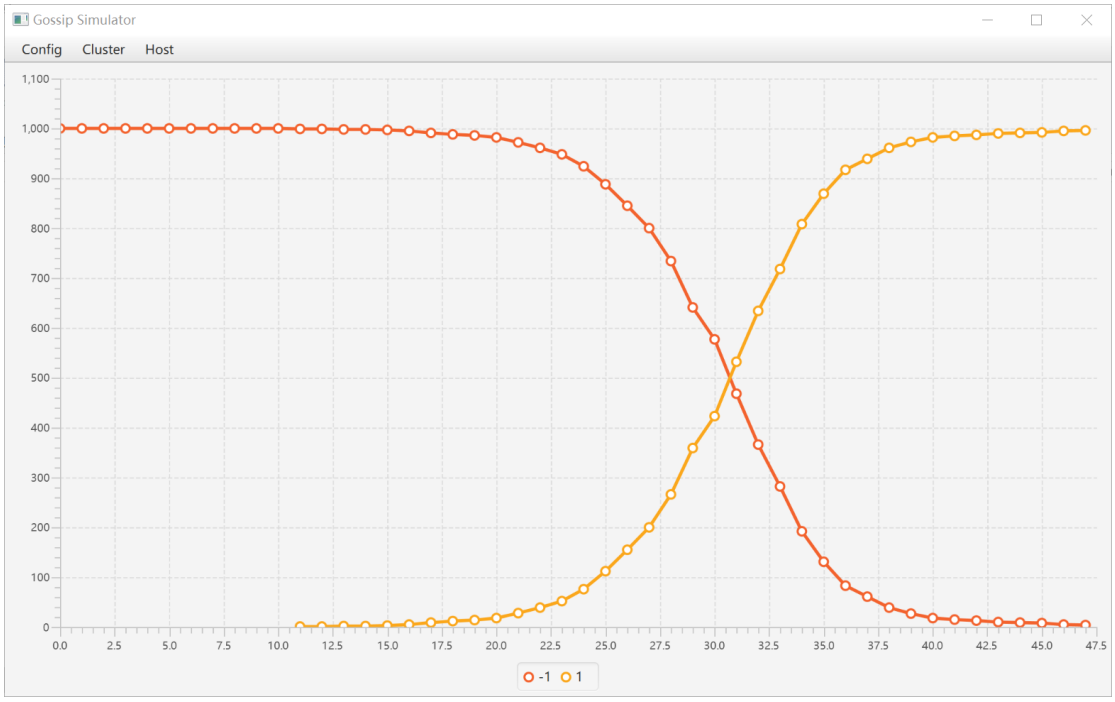

Figure3. the dynamic line chart

2)Cluster

By click the *init* in *Cluster* menu, the cluster is created and all the hosts contains the message number -1 which means the version of the message. IF the cluster is running or freezing, init will clear all the data and is ready to run.

By click the *AddMsg* you can add new message to the cluster before or after the cluster run. The different message is represented by line in different color.

By click *run* you can let the cluster start to run. And after Init or Freeze the cluster, Run is necessary to make cluster alive.

By clikc *Freeze* you can freeze the cluster that is make the God stopping send message.

By click *Restart* you can init and run the cluster.

3)Hosts

By click *destroy hosts* you are able to make some hosts cannot sending and receiving message. The input must be an integer which means how many hosts you want to make crashed. If it is negative or zero, it will be ignored. If it exceed the alive hosts, all the hosts in the cluster will be destroyed.

By click *revive hosts* you can revive destroyed hosts. If it is negative or zero, it will be ignored. If it exceeds the dead hosts, all the hosts in the cluster will be alive.

By click *add hosts* you can add more hosts to the cluster which result in a larger cluster and at the same time the sparsity of the cluster will change because it should make the topology is connected.

4)Config

By click *save* you can save the present configuration to the file.

By click *load* you can load the present configuration to the file.

By click *Config* you can create a new cluster, figure4 shows how to create a new cluster. After press the Button OK, if the window disappears, the input is legal and a new cluster is created. Otherwise the input is illegal and you should try again. Empty input means use default parameter.

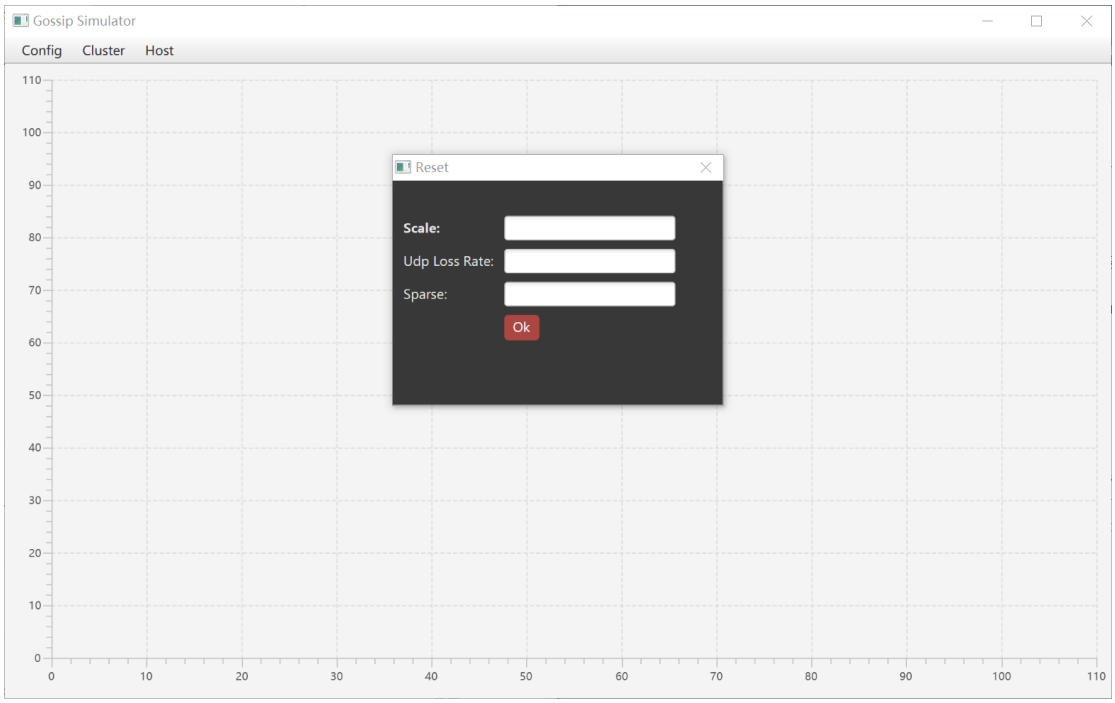

Figure4. create a new cluster

Conclusion

We first achieve a real small-scale cluster to implement gossip protocol. Then we simulate a large-scale cluster in one machine to implement gossip protocol and shows the result on graphical interface. The code is in the link https://github.com/jiangchengyike/gossip