An introduction to **PeerSim**

University of Bologna

nicolas.lazzari2@studio.unibo.it

References

- http://peersim.sf.net/
- http://peersim.sourceforge.net/doc/index.html
- http://peersim.sourceforge.net/tutorial1/tutorial1.pdf

The slides are based on the slides from previous tutors of the course.

PeerSim?

PeerSim is an open source **P2P systems simulator** developed at the Department of Computer Science, University of Bologna.

Its aim is to cope with P2P systems properties, providing high scalability.

The simulation engine

Cycle Driven Engine (CD)

The engine avoid the simulation of the transport layer. Nodes are directly connected and their actions are executed sequentially..

Event Driven Engine (ED)

Message based with aunthentic simulation of transport layers.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

Creating a simulation

Nodes, **protocols**, and **controls** are modeled as **interfaces**. The programmer has to create classes that implement such interfaces.

Node is a container of protocols. It essentially provides access to the protocol it holds.

Creating a simulation

Nodes, **protocols**, and **controls** are modeled as **interfaces**. The programmer has to create classes that implement such interfaces.

Node is a container of protocols. It essentially provides access to the protocol it holds.

CDProtocol specifies the action each node has to perform at each cycle in the Cycle-Driven model. A special kind of protocol is the **Linkable** interface, that handles neighbors of a node.

Creating a simulation

Nodes, **protocols**, and **controls** are modeled as **interfaces**. The programmer has to create classes that implement such interfaces.

Node is a container of protocols. It essentially provides access to the protocol it holds.

CDProtocol specifies the action each node has to perform at each cycle in the Cycle-Driven model. A special kind of protocol is the **Linkable** interface, that handles neighbors of a node.

Control observes and modifies the network. It collects observations on the simulation.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

A network is a set of nodes. Each node consists of a list of protocols.

Neighbors of a node are specified as a special protocol, **linkable**, and are influenced by the network structure (*fixed, newscast*).

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

Each node exposes very limited features. It is characterized by an **address** and is composed of a list of **protocols**, that actually handles its behaviour.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

The **Node** interface exposes methods such as:

```
long getID()
int getIndex()
Object clone()
Protocol getProtocol(int i)
int protocolSize()
```

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

A protocol is defined by the interface **CDProtocol**.

It is used to instruct nodes on the action they should perform. Each node can run more than one protocol, in that case they are executed **sequentially**.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

CDProtocol exposes the method

```
Object clone()
```

that is used to instantiate new nodes. In PeerSim only the first node is created using the constructor. The other ones are made by *cloning* the first node.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

The method

```
void nextCycle(Node node, int pid)
```

is called by the engine once in each cycle. The *node* parameter refers to which is the node that the protocol will be run on and *pid* refers to the id of the current protocol in the protocol array of the node.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

The **Linkable** interface is used to determine the network structure. It exposes the methods

```
int degree()
Node getNeighbor(int i)
boolean addNeighbor(Node n)
boolean contains(Node n)
```

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

See **IdleProtocol** for an example of a linkable protocol that just stores links. Useful to model static-link structures. Or **SimpleNewscast** to implement a newscast structure.

Creating a simulation

The protocol **AverageFunction** is a protocol built-in in PeerSim. When a pair of nodes interact, their values are averaged.

Creating a simulation

```java
public void nextCycle( Node node, int pid ){
    Linkable link = (Linkable) node.getProtocol( FastConfig.getLinkable(pid) );

    if (link.degree() > 0) {
        Node peer = link.getNeighbor(CommonState.r.nextInt(link.degree()));
        AverageFunction neighbor = (AverageFunction) peer.getProtocol(pid);
        double mean = (this.value + neighbor.value) / 2;
        this.value = mean;
        neighbor.value = mean;
    }
}
```

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

Controls are used to define operations that require **global knowledge** of the network. That is, to accomplish tasks such as initializing the topology of the network, adding or removing nodes, aggregating values, collecting statistics and so on.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

A control interface exposes the method

```
boolean execute()
```

that is used to implement the control logic.

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

Controls are divided into 3 categories:

- **initializers**, that are executed at the beginning of the simulation
- **dynamics**, that are executed periodically to update the network structure
- **observers**, that are executed periodically to collect information from the network

Initializer example

```java
public class PeakDistributionInitializer implements Control {
  private static final String PAR_VALUE = "value";
  private final double value;

  private static final String PAR_PROT = "protocol";
  private final int pid;

  public PeakDistributionInitializer(String prefix) {
    value = Configuration.getDouble(prefix + "." + PAR_VALUE);
    pid = Configuration.getPid(prefix + "." + PAR_PROT);
  }
```

Initializer example

```
public boolean execute() {
  for (int i = 0; i < Network.size(); i++) {
    SingleValue prot = (SingleValue) Network.get(i).getProtocol(pid);
    prot.setValue(0);
  }

  SingleValue prot = (SingleValue) Network.get(0).getProtocol(pid);
  prot.setValue(value);
  return false;
}
```

Observer example

```
public class AverageObserver implements Control
  public static final String PAR_ACCURACY = "accuracy";
  private final double accuracy;

  public static final String PAR_PROTID = "protocol";
  private final int pid;

  public AverageObserver(String name){
    this.name = name;
    accuracy = Configuration.getDouble(name+"."+PAR_ACCURACY,-1);
    pid = Configuration.getPid(name+"."+PAR_PROTID);
  }
```

Observer example

```
public boolean execute() {
  long time = peersim.core.CommonState.getTime();
  IncrementalStats stats = new IncrementalStats();

  for (int i = 0; i < len; i++) {
    SingleValue protocol = (SingleValue) Network.get(i).getProtocol(pid);
    stats.add(protocol.getValue());
  }

  System.out.println(name + ":" + time + " " + stats);
  return (stats.getStD() <= accuracy);
}
```

Creating a simulation

1. Create a **network** of **nodes**
2. Choose some **protocols**
3. Choose **controls**
4. Simulate

Simulation is performed by executing sequentially the protocol of each node.

Simulation workflow

```
for i in simulation.experiments:
    create Network
    createPrototype Node # creates all the inner protocols
    for j in network.size:
        clone prototype node into Network
    CreateControls(initializers, dynamics, observers)
    for k in simulation.cycles:
        for j in network.size:
            for p in protocols:
                Network.get(j).getProtocol(p).nextCycle()
        execute controls
        if (one control returned True) break
```

Setting up the simulation

In PeerSim the size of the network, the protocols to use, the controls and so on are configured by using a **configuration file**.

It is a plain text file containing key-value pairs.

The configuration is made of three parts: **general setup**, **protocols definition**, **controls definition**.

General setup

```
simulation.cycles 30 # nr of simulation cycles

control.shf Shuffle # shuffle the order nodes are visited in

network.size 50000 # size of the network
```

Protocols definition

```
protocol.lnk IdleProtocol # Linkable structure of the network

protocol.avg example.aggregation.AverageFunction # the actual protocol
protocol.avg.linkable lnk # instruct the protocol to use the defined linkable
```

Controls definition

```
# Wire each node to k randomly chosen nodes
init.rnd WireKOut
init.rnd.protocol lnk
init.rnd.k 20

# Setup two initializers
init.peak example.aggregation.PeakDistributionInitializer
init.peak.value 10000
init.peak.protocol avg

init.lin LinearDistribution
init.lin.protocol avg
init.lin.max 100
init.lin.min 1
```

Controls definition

```
include.init rnd lin # select the initializer (lin in this case, bu peak can
                     # be used)

control.avgo example.aggregation.AverageObserver # Set the observer
control.avgo.protocol
```

Execute the simultion

```
java -cp "peersim-1.0.5.jar:jep-2.3.0.jar:djep-1.0.0.jar" peersim.Simulator conf.txt
```