Simulating overlay networks with PeerSim

Moreno Marzolla
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna

http://www.moreno.marzolla.name/
Acknowledgements

- These slides are based on those kindly provided by Andrea Marcozzi
- Part of this material is taken from the presentation “PeerSim: Informal Introduction” by Alberto Montresor and Gianluca Ciccarelli
What is PeerSim?

- PeerSim is an open source P2P systems simulator developed at the Department of Computer Science, University of Bologna
- It has been developed with Java
- Available on Source Forge (http://peersim.sf.net)
- Its aim is to cope with P2P systems properties
- High Scalability (up to 1 million nodes)
- Highly configurable
- Architecture based on pluggable components
The peersim Simulation Engine

- **Cycle-Driven (CD)**
  - Quick and dirty: no messages, no transport, synchronized
  - Specialized for epidemic protocols
  - Tested up to $10^7$ nodes

- **Event-Driven (ED)**
  - More realistic: message-based, realistic transports
  - Tested up to $2.5 \times 10^5$ nodes
  - Not considered in these lectures
Peersim

Simulation Engine

Initializers

init.traces
init.random

Observers

control.observe-tman
control.observe-slice

Dynamics

control.churn

Network

node 1
node 2
node 3
...
node n

Protocol vector

protocol.transport
Prot. 1
Prot. 1
Prot. 1
...
Prot. 1

protocol.newscast
Prot. 2
Prot. 2
Prot. 2
...
Prot. 2

protocol.tman
Prot. 3
Prot. 3
Prot. 3
...
Prot. 3

Protocol names
Network Representation

- **Network**
  - global array which contains all the network nodes

- **Node**
  - each node’s state and actions are described through a stack of protocols. Protocols are accessed through a Pid

- **Linkable**
  - interface used to access and manage node’s view and other properties of a node
  - More on the Linkable interface shortly...
Network Representation

- **CDProtocol**: interface used to describe node’s actions at each cycle.
  - A generic node can both perform local actions (CDProtocol) and manage the local view (Linkable)
- **Control**: performs the global initialization and performance analysis
  - Initializers are just Control objects with the peculiarity of being executed just once at the beginning
public interface Node extends Fallible, Cloneable
{

/**
 * Returns the <code>i</code>-th protocol in this node. If <code>i</code>
 * is not a valid protocol id (negative or larger than or equal to the number
 * of protocols), then it throws IndexOutOfBoundsException.
 */
public Protocol getProtocol(int i);

/**
 * Returns the number of protocols included in this node.
 */
public int protocolSize();

/**
 * Returns the unique ID of the node. It is guaranteed that the ID is unique
 * during the entire simulation, that is, there will be no different Node
 * objects with the same ID in the system during one invocation of the JVM.
 * Preferably nodes
 * should implement <code>hashCode()</code> based on this ID.
 */
public long getID();

/* ... */
}
public interface Protocol extends Cloneable
{

/**
 * Returns a clone of the protocol. It is important to pay attention to
 * implement this carefully because in peersim all nodes are generated by
 * cloning except a prototype node. That is, the constructor of protocols is
 * used only to construct the prototype. Initialization can be done
 * via {link Control}s.
 */
public Object clone();
}
Main Interfaces: Protocol

- The *CDProtocol* interface is used to define cycle-driven protocols, that is the actions performed by each *node* at each simulation cycle.
- Each node can run more than one protocol.
- Protocols are executed sequentially.
/**
* Defines **cycle driven protocols**, that is, protocols that have a periodic
* activity in regular time intervals.
*/

public interface CDProtocol extends Protocol {

/**
 * A protocol which is defined by performing an algorithm in more or less
 * regular periodic intervals.
 * This method is called by the simulator engine once in each cycle with
 * the appropriate parameters.
 *
 * @param node
 * the node on which this component is run
 * @param protocolID
 * the id of this protocol in the protocol array
 */

public void nextCycle(Node node, int protocolID);
}
Main Interfaces: Linkable

- *Linkable* is used to manage node’s view.
- Typical actions are:
  - *Add* neighbour
  - *Get* neighbour
  - Node’s *degree*
- Note: the *Linkable* interface does **not** support nodes removal; you need to define your own interface to do so
public interface Linkable extends Cleanable {
    /**
     * Returns the size of the neighbor list.
     * /
    public int degree();

    /**
     * Returns the neighbor with the given index.
     * /
    public Node getNeighbor(int i);

    /**
     * Add a neighbor to the current set of neighbors.
     * /
    public boolean addNeighbor(Node neighbour);

    /**
     * Returns true if the given node is a member of the neighbor set.
     * /
    public boolean contains(Node neighbor);

    /**
     * A possibility for optimization. An implementation should try to
     * compress its internal representation. Normally this is called
     * by initializers or other components when
     * no increase in the expected size of the neighborhood can be
     * expected.
     * /
    public void pack();
}
The Linkable interface
The Control interface

- Used to define operations that require global network knowledge and management, such as:
  - **Initializers**, executed at the beginning of the simulation
    - Initial topology
    - Nodes state
  - **Dynamics**, executed periodically during the simulation
    - Adding nodes
    - Removing nodes
    - Resetting nodes
  - **Observers**, executed periodically during the simulation
    - Aggregated values from all the nodes
The Control interface

/**
 * Generic interface for classes that are responsible for observing or modifying
 * the ongoing simulation. It is designed to allow maximal flexibility therefore
 * poses virtually no restrictions on the implementation.
 */

public interface Control
{

/**
 * Performs arbitrary modifications or reports arbitrary information over the
 * components.
 * @return true if the simulation has to be stopped, false otherwise.
 */
public boolean execute();
}
for i := 1 to simulation.experiments do
create Network
create prototype Node :
    for i := 1 to #protocols do
        create protocol instance
    for j := 1 to network.size do
        clone prototype Node into Network
create controls ( initializers, dynamics, observers )
execute initializers
for k := 1 to simulation.cycles do
    for j := 1 to network.size do
        for p := 1 to #protocols do
            execute Network.get(j).getProtocol(p).nextCycle()
        execute controls
    if ( one control returned true ) then
        break
CDSimulator
Peersim Configuration

- Once all components have been implemented the whole simulation has to be set up
  - Declare what components to use
  - Define the way they should interact
- In Peersim simulations are defined through a plain text configuration file
- Configuration file is divided in 3 main parts
  - General setup
  - Protocol definition
  - Control definition
Peersim Configuration

- Plain ASCII file containing key-value pairs
  - Lines starting by # are ignored
- Syntax:

  \{{protocol,init,control}\}.string_id
  [full_path]classname

- The class Initializer implements the interface Control
- An Initializer object is run at the beginning of the simulation
Peersim Configuration

- Component parameters’ syntax

\{\text{protocol}, \text{init}, \text{control}\}.\text{string\_id. parameter\_name parameter\_value}

Must have been previously defined
Example

random.seed 1234567891

control.shf Shuffle

simulation.cycles 100
simulation.experiments 50

network.size 10^6
network.node peersim.core.GeneralNode

Global property: used to initialize the RNG

Shuffles the order in which nodes are visited at each cycle

Maximum number of simulation cycles

Number of nodes in the network
V-MAN
Goal

- Reduce power consumption in Cloud infrastructures

Idea

- Migrate Virtual Machines away from lightly loaded servers
- Servers running no VM can be put to sleep
V-MAN—Example

(a) Before consolidation

(b) After consolidation of VM₁ and VM₄ to host 2
V-MAN—Example

- Server consolidation is implemented using a simple gossip protocol (a variant of aggregation)

I have 2 VMs

I have only one

Ok, I send one VM to you
V-MAN

- Each node $i$ maintains the number $H_i$ of VM it is currently running
  - All nodes have a maximum capacity $C$
- Node $i$ select random peer $j$:
  - If $H_i \leq H_j \rightarrow$ node $i$ sends its VMs to node $j$
  - If $H_i > H_j \rightarrow$ node $i$ receives VMs from node $j$
RandomDistributionInitializer

- Initially, each node is assigned a random number of VMs
- This is done by the RandomDistributionInitializer
public class RandomDistributionInitializer implements Control, NodeInitializer {

    // ... constants and local variables omitted ...

    /**
     * This class provides a simple random distribution in a bounded
     * interval defined by parameters {@link #PAR_MIN} and {@link #PAR_MAX}.
     */
    public boolean execute() {
        int tmp;
        for (int i = 0; i < Network.size(); ++i) {
            initialize( Network.get(i) );
        }
        return false;
    }

    /**
     * Initialize a single node by allocating a random number of virtual
     * machines, defined by parameters {@link #PAR_MIN} and {@link #PAR_MAX}.
     */
    public void initialize(Node n) {
        int tmp = min + CommonState.r.nextInt( max-min+1 );
        ((SingleValue) n.getProtocol(protocolID)).setValue(tmp);
    }
}
BasicConsolidation

- This class implements the CDProtocol interface
  - V-MAN is actually implemented here
- BasicConsolidation also inherits from SingleValueHolder
  - The value held in each node is the current number of running VMs
public class BasicConsolidation extends SingleValueHolder implements CDProtocol {

    /**
    * Node capacity. The capacity is the maximum number of
    * Virtual Machines that a node can host. Defaults to 1.
    *
    * @config
    */
    protected static final String PAR_CAPACITY = "capacity";

    /** Capacity. Obtained from config property \{@link #PAR_CAPACITY\}. */
    private final int capacity_value;

    /**
    * Standard constructor that reads the configuration parameters. Invoked by
    * the simulation engine.
    *
    * @param prefix
    * the configuration prefix for this class.
    */
    public BasicConsolidation(String prefix) {
        super(prefix);
        // get capacity value from the config file. Default 1.
        capacity_value = (Configuration.getInt(prefix + "." + PAR_CAPACITY, 1));
    }
}
/**
 * Using an underlying {@link Linkable} protocol
 * performs a consolidation step with all neighbors of the node
 * passed as parameter.
 *
 * @param node
 * the node on which this component is run.
 * @param protocolID
 * the id of this protocol in the protocol array.
 */

public void nextCycle(Node node, int protocolID) {
    int linkableID = FastConfig.getLinkable(protocolID);
    Linkable linkable = (Linkable) node.getProtocol(linkableID);

    for (int i = 0; i < linkable.degree(); ++i) {
        Node peer = linkable.getNeighbor(i);
        // The selected peer could be inactive
        if (!peer.isUp())
            continue;
        BasicConsolidation n = (BasicConsolidation) peer.getProtocol(protocolID);
        doTransfer(n);
    }
}
/**
 * Performs the actual consolidation selecting to make a PUSH or PULL
 * approach. The idea is to send the maximum number of VMs from
 * the node with fewer VMs to the other one.
 *
 * @param neighbor
 *            the selected node to talk with.
 */

protected void doTransfer(BasicConsolidation neighbor) {
    int a1 = (int)this.value;
    int a2 = (int)neighbor.value;
    if (a1 == 0 || a2 == 0) return;
    int a1_avail = capacity_value - a1;
    int a2_avail = neighbor.capacity_value - a2;
    int trans = Math.min( Math.min(a1,a2),
                          Math.min(a1_avail, a2_avail) );

    if (a1 <= a2) {
        // PUSH
        a1 -= trans;
        a2 += trans;
    } else {
        // PULL
        a1 += trans;
        a2 -= trans;
    }
    assert(a1 >= 0 && a1 <= capacity_value);
    assert(a2 >= 0 && a1 <= capacity_value);
    this.value = (float)a1;
    neighbor.value = (float)a2;
}
VMObserver

- This class implements the Control interface
- It is used to print to standard output the number of servers with exactly $k$ VMs, for all $k = 0, \ldots, C$
public class VMObserver implements Control {

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

    public VMObserver(String name) {
        this.name = name;
        pid = Configuration.getPid(name + "." + PAR_PROT);
    }

    public boolean execute() {
        IncrementalFreq freqs = new IncrementalFreq();
        long time = peersim.core.CommonState.getTime();
        int capacity = 0;
        for (int i = 0; i < Network.size(); i++) {
            BasicConsolidation protocol = (BasicConsolidation) Network.get(i).getProtocol(pid);
            capacity = protocol.getCapacity();
            freqs.add((int)protocol.getValue());
        }
        System.out.print(name + ": " + time);
        for (int j = 0; j <= capacity; ++j )
            System.out.print(" " + freqs.getFreq(j));
        System.out.println();
        return false;
    }
}
Configuration file

Simulation settings

- simulation.cycles 20
- simulation.experiments 10
- network.size 10000
- WIRED 20
- CORES 8

Random seed

- random.seed 1234567890

Protocols settings

- protocol.lnk example.newscast.SimpleNewscast
- protocol.lnk.cache WIRED
- protocol.vman example.vman.BasicConsolidation
- protocol.vman.capacity CORES
- protocol.vman.linkable lnk

Initialization settings

- init.rnd WireKOut
- init.rnd.protocol lnk
- init.rnd.k WIRED

- init.ld example.vman.RandomDistributionInitializer
- init.ld.protocol vman
- init.ld.min 0
- init.ld.max CORES

Include settings

- include.init rnd ld

Control settings

- control.shf Shuffle
- control.vmo example.vman.VMObserver
- control.vmo.protocol vman
Config File: General Settings

- Each simulation is executed for 20 steps
- Perform 10 independent simulation runs
- The network has 10000 nodes
- WIREK and CORES are constants
- The seed for the random number generator is initialized with a specific value (otherwise, it is initialized with a random number)

```plaintext
simulation.cycles 20
simulation.experiments 10
network.size 10000
WIREK 20
CORES 8
random.seed 1234567890
```
Protocol “\textit{lnk}” is NewsCast
The view size is set to the constant WIREK
Protocol “\textit{vman}” is V-MAN
Each node can support at most CORES VMs
The local view is maintained by protocol “\textit{lnk}”
Newscast

- Each node maintains a *local view*
  - Set of nodes it is connected to
- At each step, neighbors exchange their local views
- Each node merges the remote view with its local one, discarding old entries and keeping fresh ones
Newscast
Newscast

\{1, 3, 5, 6\}
Initially, build a random graph by wiring each node to \textit{WIREK} other random nodes.

RandomDistributionInitializer assigns a random number in \([0, \ldots, \text{CORES}]\) of VMs to each node.

```sh
init.rnd WireKOut
init.rnd.protocol lnk
init.rnd.k WIREK

init.ld example.vman.RandomDistributionInitializer
init.ld.protocol vman
init.ld.min 0
init.ld.max CORES

include.init rnd ld

control.shf Shuffle
control.vmo example.vman.VMObserver
control.vmo.protocol vman
```
Additional resources

- PeerSim Web Page
  http://peersim.sf.net/

- Class documentation

- Tutorial for the Cycle-based engine