LSim: test and evaluation of distributed applications in realistic environments

Manel Pérez <mpeerez@uoc.edu>
Joan Manel Marquès <jmarquesp@uoc.edu>
DPCS research group @ UOC

Outline

- Distributed applications development problem
- Easing the problem with LSim
- LSim main features
- Current and future goals
- Conclusions
- Links and references

- Distributed applications development problem
- Easing the problem with LSim
- LSim main features
- Current and future goals
- Conclusions
- Links and references

Distributed applications development problem

- Software testing is hard and time consuming, distributed applications testing is even more hard
- Realistic environments problem:
  - Distributes nodes over a network like internet implies non-deterministic behaviour, non-reproducibility
  - Complex timing of events, complex states
  - Setting up a realistic environment manually is complex and requires time

Distributed applications development problem

- Different approaches to distributed applications testing
  - **Simulation**
    Allows to run large-scale tests that are difficult to do on other ways. Great control but hard to set up and simulate realistic conditions
  - **Emulation**
    Easy to abstract different components and analyse their behaviour, tests are reproducible. Not a real network. Hard to test extreme conditions achieved with simulation.
  - **Real deployment** on a distributed environment
    Test components and whole application under realistic conditions. Hard to set up environment and large-scale tests.

- Distributed applications development problem
- Easing the problem with LSim
- LSim main features
- Current and future goals
- Conclusions
- Links and references

Easing the problem with LSim

- LSim helps developers to test the application in a real distributed environment:

Deployment

  - Automates the deployment of the application components
  - Flexible parametrization and initialization of tests

Coordination

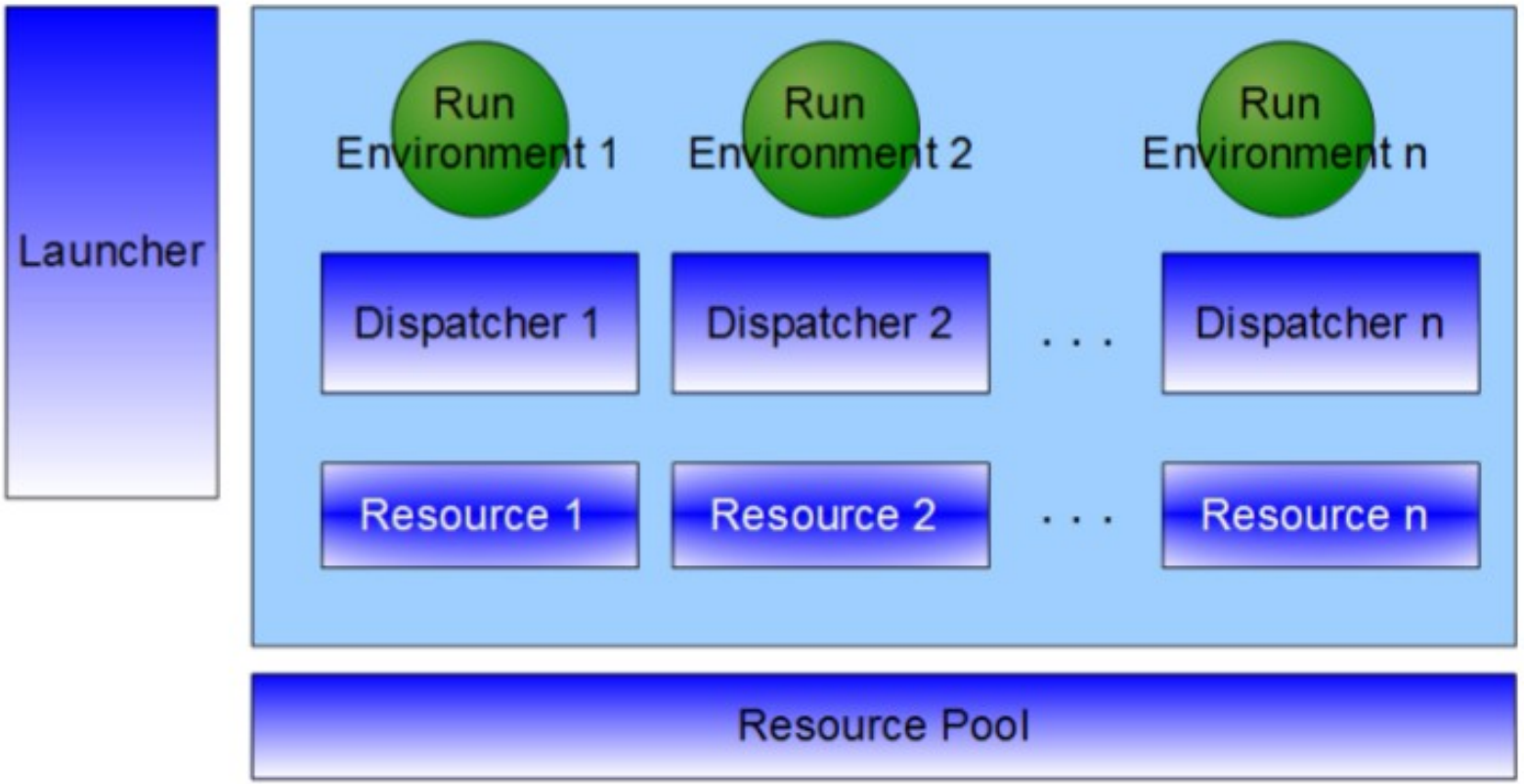
  - Coordinates the execution of the components of the distributed application
  - Collects and evaluates results
  - Collects exceptions from different running instances

- Distributed applications development problem
- Easing the problem with LSim
- LSim main features
- Current and future goals
- Conclusions
- Links and references

LSim main features

- LSim is composed by two main elements:
  - **LSim framework**:
    Automatically deploys the application or protocol in the resources that will perform the tests
  - **LSim library**
    Automates the implementation, coordination and collection of results
- Currently implemented in Java and run applications inside a JVM

LSim framework



Running environment: environment that runs the application instance.
Dispatcher: receives requests to execute an application instance and runs it in the running environment. It has control over the application, being able to stop it if needed.
Resource: computer contributed to the community network that runs a dispatcher.
Resource-pool: Allows the discovering of available resources, i.e. the location (IP address ad port) of dispatchers running in nodes contributed to the framework.
Launcher: selects the nodes to participate in an experiment according to the requirements included in a descriptor and deploys the application of protocol instances in these nodes.

LSim library

- Tries to reduce the burden of giving initialization values to each instance, to coordinate the start of each instance, to collect results, to synchronize intermediate points, etc.
- **Objective**: to be as **less intrusive as possible** to the original code of the application or protocol
- Has 3 different **roles**, each one being responsible of a specific functionality related to the execution of an experiment:
  - **Coordinator**: coordinates the different instances that run the application
  - **Worker**: runs the application itself
  - **Evaluator**: collects results from workers and evaluates them

LSim library API

Method	Coordinator	Worker	Evaluator
init(Handler)	Initializes itself and generates configuration information for workers and evaluators	Initialize the component. Blocks until it receives the initial configuration and parameters from the coordinator	
start(Handler)	Generates start information for workers and sends them a start signal	Starts the component. Blocks until the coordinator notifies it to start the experiment.	
sendResult(Handler, evalId)	*Not applicable*	Sends a result and continue its execution. Results will be send to *evalId* Evaluator	Gets results from workers.
synchronize(Handler, label)	*Not applicable*	Blocks on the specified label until the Coordinator notifies that execution can continue	*Not applicable*
stop(Handler)	Waits for all workers to stop and then finishes.	Ends the execution of the component	
sendException(Object)	After capturing a java exception, send an object to a given location. Application developers can customize it's behavior		

Adapting an application to use LSim library

Application code before adapting to LSim

```
public static void main(String args[])throws Exception{

    int num = Integer.valueOf(args[0]);
    List<InetSocketAddress> brothers = getBrothers();
    int port = Integer.valueOf(args[1]);

    SyncBuffer buffer=new SyncBuffer();
    ListenerPlus list=new ListenerPlus(port,buffer);

    list.bindConnection();
    for(int i=0; i< brothers.size(); ++i){
        Net.send(num, brothers.get(i));
    }
    int n_mess = brothers.size()-1;
    int mess = 0;
    int sum = num;
    while(mess < n_mess){
        while(!buffer.isEmpty()){
            sum += Integer.valueOf(buffer.getObject().toString());
            mess++;
        }
        if(mess < n_mess){
            buffer.StartWait();
        }
    }
    System.out.println("Result "+sum);
    Net.send(new Result(String.valueOf(num),sum), getCorrector());
}
```

Application Initialization

Test Initialization

Application Execution

Finish Execution

Application code adapted to LSim

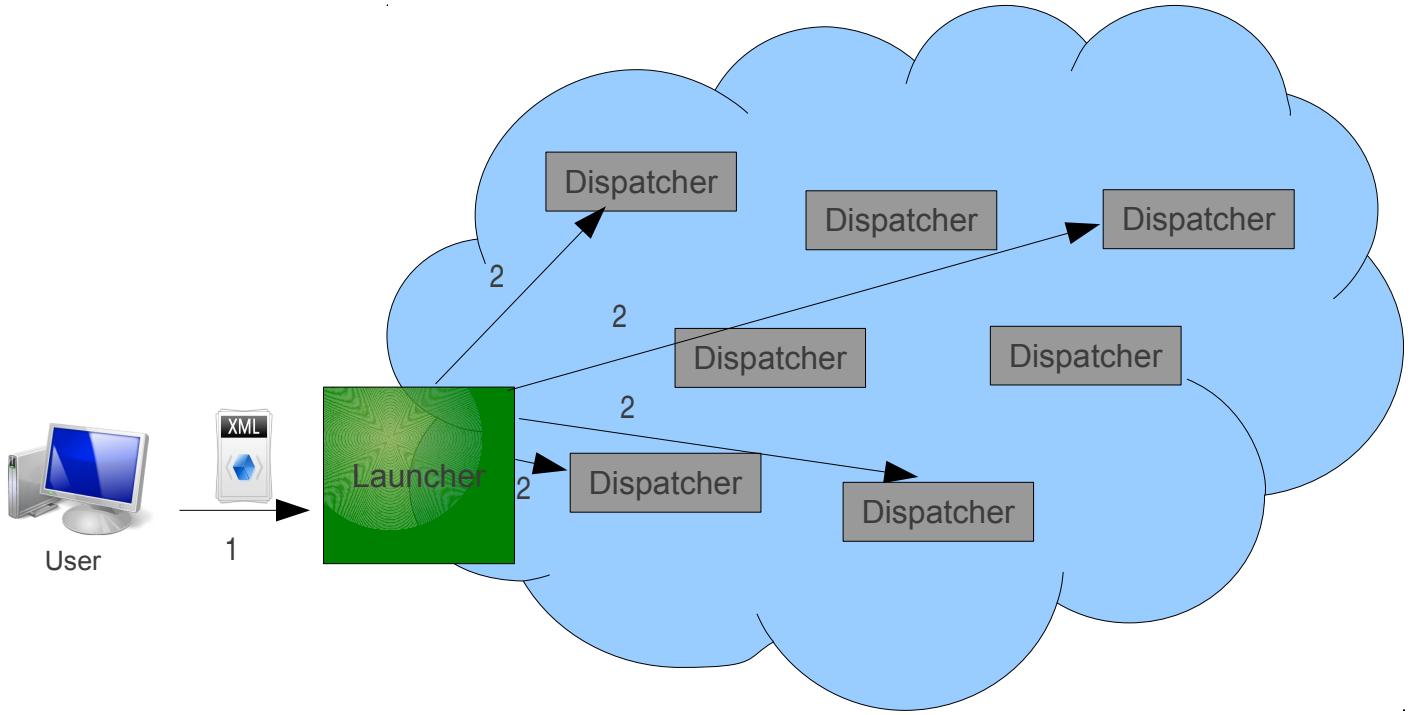
```
public void start(LSimDispatcherHandler disp) {
    LSimWorker lsim = new LSimWorker();
    lsim.setDispatcher(disp);
    SyncBuffer buffer = new SyncBuffer();
    ListenerPlus list = new ListenerPlus(0,buffer);

    PlusInitHandler p = new PlusInitHandler(lsim, 10);
    p.setParams(list);
    lsim.init(p);

    int num=p.number;
    List<InetSocketAddress> brothers=p.others;

    lsim.start(new DummyHandler());

    for(int i=0; i< brothers.size(); ++i){
        Net.send(num, brothers.get(i));
    }
    int n_mess = brothers.size()-1;
    int mess = 0;
    int sum = num;
    while(mess < n_mess){
        while(!buffer.isEmpty()){
            sum += Integer.valueOf(buffer.getObject().toString());
            mess++;
        }
        if(mess < n_mess){
            buffer.StartWait();
        }
    }

    System.out.println("Result "+sum);
    lsim.sendResults(new ResultHandler(new Result(String.valueOf(num), sum)));
}
```

init( )

start( )

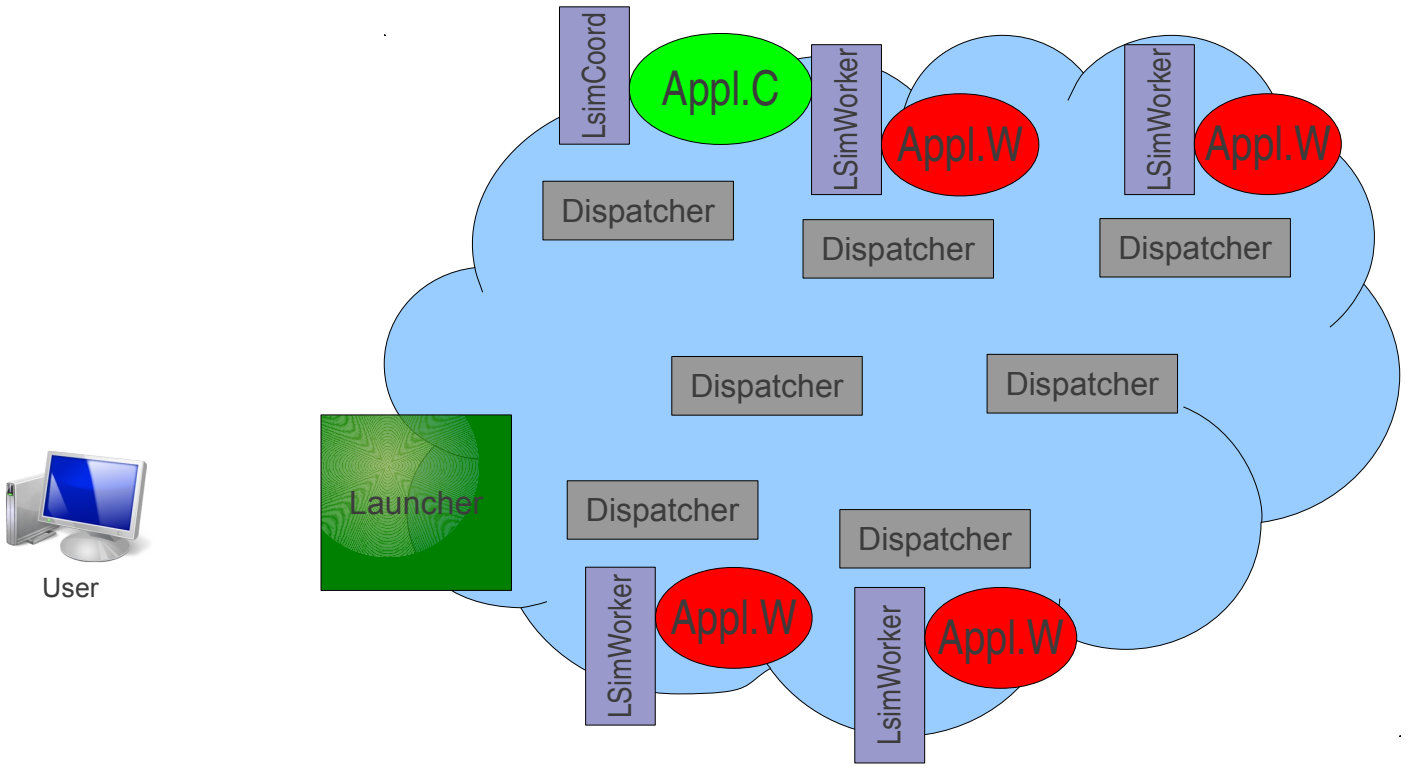
sendResult( )

Launching an experiment



1. User sends specification to Launcher
2. Launcher locate resources and deploy needed instances

Launching an experiment



3. Instances deployed and ready

- Distributed applications development problem
- Easing the problem with LSim
- LSim main features
- Current and future goals
- Conclusions
- Links and references

Current and future goals

- Make public available current version so users can run their experiments:
  - Automatically deploy experiments to available dispatchers
  - Develop a web interface and desktop client to simplify experiment launching and execution tracking
- Deploy dispatchers to multiple machines including students machines
- Use LSim to evaluate the practices of students of Distributed Systems subject next course

Current and future goals

- Add synchronization methods so developers can test more complicated states of their applications
- Some examples:

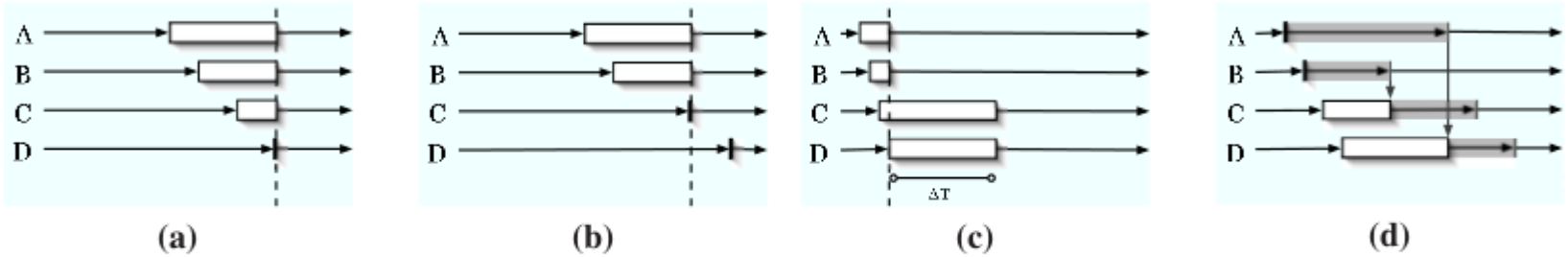


Figure 1: (a) Traditional semantics: All hosts enter the barrier (indicated by the white boxes) and are simultaneously released (indicated by dotted line). (b) Early entry: The barrier fires after 75% of the hosts arrive. (c) Throttled release: Hosts are released in pairs every ΔT seconds. (d) Counting semaphore: No more than 2 processes are simultaneously allowed into a "critical section" (indicated by the grey regions). When one node exits the critical section, another host is allowed to enter.

Jeannie Albrecht, Christopher Tuttle, Alex C. Snoeren, and Amin Vahdat. Loose Synchronization for Large-Scale Networked Systems. In USENIX Annual Technical Conference (USENIX), June 2006.

- Distributed applications development problem
- Easing the problem with LSim
- LSim main features
- Current and future goals
- Conclusions
- Links and references

Conclusions

- We presented LSim, a tool to help developing, testing and evaluation of distributed applications
- We are almost ready to make public available a version of the tool and get plenty of feedback from current distributed application developers.

- Distributed applications development problem
- Easing the problem with LSim
- LSim main features
- Current and future goals
- Conclusions
- Links and references

Links and references

- LSim page with documentation and tutorials
  - http://dpcs.uoc.edu/projects/lsim
- Contact information:
  - mpeerez@uoc.edu
  - jmarquesp@uoc.edu

Thanks for you attention