UNIVERSITAT OBERTA DE CATALUNYA

LSim overview

Joan Manuel Marquès
Manel Pérez
Esteve Verdura

DPCS Research group (http://dpcs.uoc.edu)
Universitat Oberta de Catalunya

Technical Report. July, 19th 2012

Abstract

In this paper we introduce LSim, a tool for testing applications or protocols in a set of distributed computers. LSim is formed by: a) a library (LSim Library) that automates the implementation, coordination and collection of results, and b) a framework (LSim Framework) that automatically deploys the application or protocol in the resources that will perform the tests. A prototype of LSim that works with java programs is currently available.

Overview

LSim is a tool for testing applications or protocols in a set of distributed computers. It is formed by: a) a library (LSim Library) that automates the implementation, coordination and collection of results, and b) a framework (LSim Framework) that automatically deploys the application or protocol in the resources that will perform the tests.

LSim might be used to easily run java applications or protocols in a community network. Our current prototype is implemented in java and run applications or protocols inside a JVM. More information about LSim project and latest implementation might be found at http://dpcs.uoc.edu/projects/lsim

*A. **Lsim Framework***

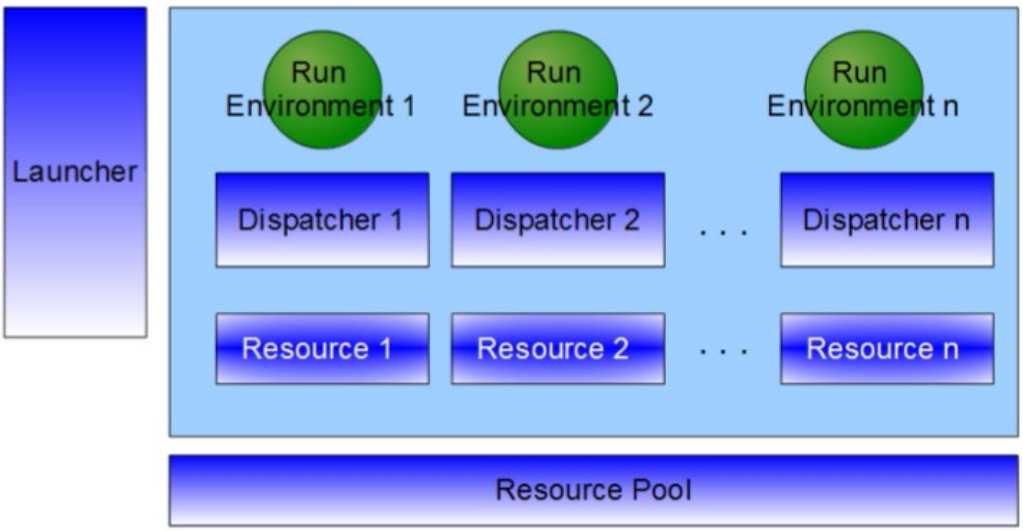

Figure 1. LSim Framework architecture.

Figure 1 details the different elements of LSim:

- **Running environment**: environment that runs the application or protocol instance. Current prototype uses a JVM but Virtual Machines or any other safer technology might be used.
- **Dispatcher**: receives requests to execute an application or protocol instance and runs it in the running environment. It has control over the application, being able to stop it if needed. Furthermore, it is responsible of all communication among LSim instances, avoiding the developer the burden of dealing with initialization and coordination tasks.
- **Resource**: computer contributed to the community network that runs a dispatcher and, therefore, is able to run LSim experiments.
- **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. We provide a centralized implementation of this resource-pool and a decentralized implementation using CoDeS. The decentralized implementation using CoDeS is more suitable for self-managed community networks.
- **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. Descriptor includes, among others, url of the repository that contains the code to be run, number of instances of each portion of code, initialization information, etc.

B. LSim Library

Once deployed, the application has to run in a coordinated manner. This is not an easy task: each application or protocol to be tested should implement all the necessary logic to know where other instances are located, to give initialization values to each instance, to coordinate the start of each instance, to collect results, to synchronize intermediate points, etc. LSim Library tries to reduce all this burden to developers by providing a library that hides most parts of these functionalities behind a reduced set of methods. Therefore, developer only has to implement the parts that are specific to its application or protocol and LSim Library will do the rest.

The main objective of LSim Library is to be as less intrusive as possible to the original code of the application or protocol. Like that, the deployed code will be almost the same than if it had been deployed in a real setting without using LSim.

1) Roles LSim Library

LSim Library has three 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 or protocol.
- **Worker**: runs the application or protocol itself.
- **Evaluator**: collects results from workers and evaluates them.

Developer has to implement a coordinator and one or more evaluator components for each experiment to deal with specificities of the experiment that is carrying out. Worker should be able to run the initial application or protocol with the minimal changes.

2) LSim library functionality

LSim offers different functions to configure the experiments and coordinate its execution: initialization, start, synchronization, send results and stop. Each method acquires a different behavior depending on the role it is running.

To make the application independent from the LSim code, LSim library offers handlers. Handlers allow application developers to personalize the code executed when any of the mentioned phases starts. Our current prototype offers simple handlers that can be used on any phase.

Table 1 describes the functionality of each method for each role.

TABLE I. LSIM 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		

3) *Handlers*

Most of the methods in table 1 use a handler (callback) passed as a parameter, what allows the developer of the application or protocol to provide the desired behavior to each method. *init*, *start* and *synchornize* methods execute the handler after unblocking and is used to get information from the *Coordinator*. *sendResult* method execute the handler in order to prepare the data that LSim has to send to the Evaluator. *stop* handler is executed before stopping the application or protocol.

Figure 2 shows an example of *initHandler*, handler for the init method. It is used to obtain the initialization parameters of the Worker. Figure 3 shows a portion of the code that has to be added to the application or protocol to obtain this initialization information.

```
public class InitHandler implements Handler{
  private List<Object> param;
  /**
  * Creates an instance of initial handler
  */
  public InitHandler(){
  }
  public Object execute(Object obj) {
    param=(List<Object>)obj;
    return null;
  }
  /**
  * Gets the parameters for the element
  * @return List of object, where every object is a parameter
  * that you should cast at correct type according to the order
  * on experiment specification.
  */
  public List<Object> getParameters(){
    return param;
  }
}
```

Figure 2. Example: handler

```
// init
InitHandler init = new InitHandler();
lsim.init(init);
// getting parametres
List<Object> param = init.getParameters();
String p1 = param.get(0).toString();
int p2 = (Integer)param.get(1);
System.out.println("Received parameters: " + p1 + " " + p2);
```

Figure 3. Example: using a handler from the application or protocol.

C. *Running an experiment using LSim*

In this subsection we show that minimal changes are required to adapt an application to be used in LSim and list the steps to deploy and run an application in a set of nodes that run a LSim dispatcher.

1) *Preparing the application*

A main objective of the design of LSim is not being intrusive (or at least minimize it) in the application or protocol code. However, there are minor changes that need to be done to the application or protocol.

The usual flow of an application or protocol is: an initialization (where configuration information is acquired), run the core of the application of protocol, and stop. In addition, when evaluating the behavior of the application or protocol it could be interesting to collect partial results (while running the core of the application or protocol) or final results (after this core part is ended and before stopping). As seen in table 1 LSim Library provides methods to do these functionalities, what makes very easy and not intrusive to adapt the application or protocol to use LSim Library. Figure 4 shows an example of adapting an application to use LSim Library.

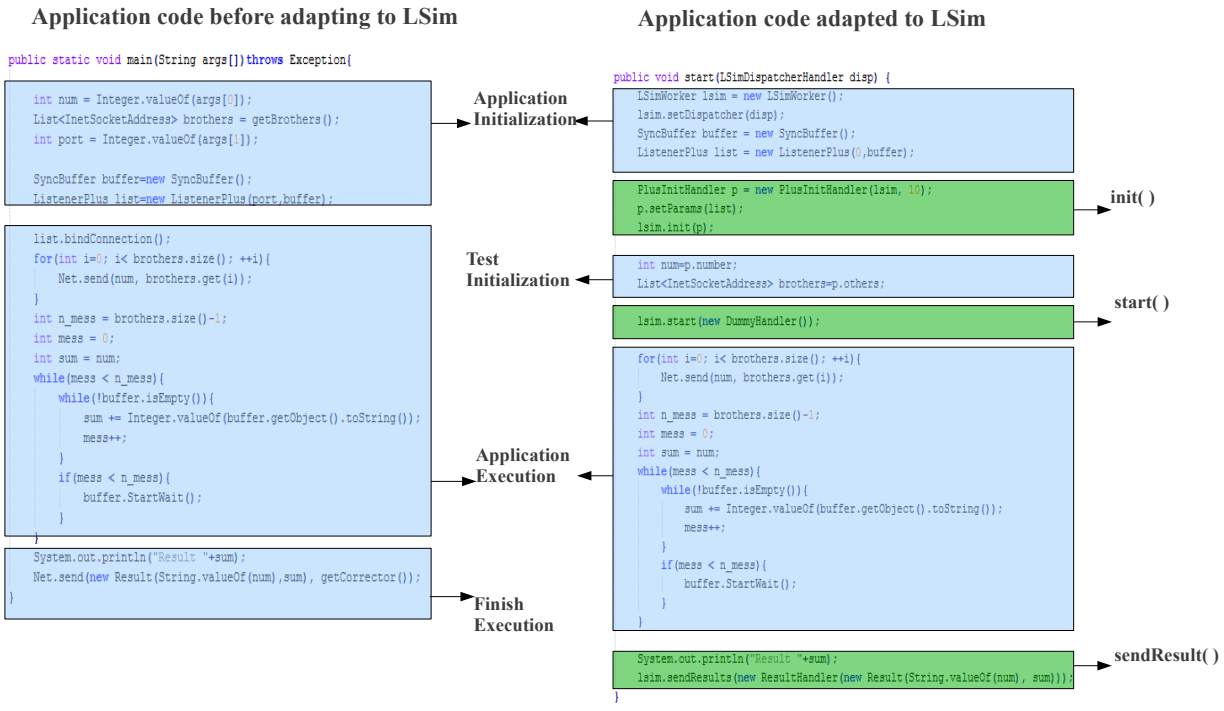

Figure 4. Example of adapting an application to use LSim Library

2) *Deploying and executing an experiment*

Steps to execute an experiment using LSim:

1. User creates an experiment descriptor and sends it to Launcher. (step 1 in figure 5)

2. Launcher selects a set of nodes that satisfy the requirements specified in the descriptor and sends them the necessary information to run a Coordinator or an instance of the application or protocol. (step 2 in figure 5)

3. Each dispatcher instantiates the *Coordinator* or the instance of the application or protocol in the running environment (figure 6)

4. *Coordinator* initializes each worker (using init method described in table 1)

5. *Coordinator* starts workers (using start method described in table 1)

6. Instances of the application or protocol communicate among them according to its internal logic. If required, they use *synchronize* method provided by LSim to synchronize the execution.

7. When a *worker* has a result (partial or final result) uses *sendResult* method provided by LSim to transparently send the result to an *Evaluator* component.

8. When a *worker* finishes or receives a stop message from the *coordinator*, it executes the stop method and finishes.

9. When an *Evaluator* has received all expected results evaluates them and sends the final result to the location indicated in the descriptor of the experiment.

During the whole experiment, if any instance of the application or protocol throws an exception, developer can capture it and use *sendException* method to send (transparently) the exception to a given location. This helps the debugging of distributed application or protocol.

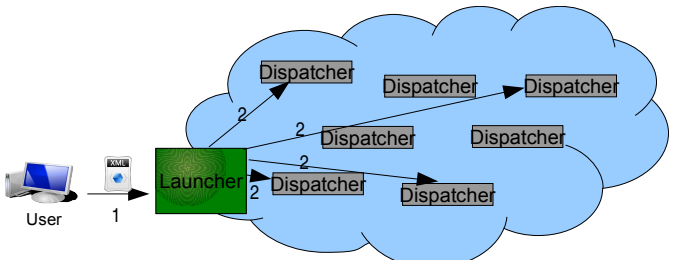

Figure 5. Steps 1 and 2 to deploy an experiment using LSim framework. *(1)* User sends the experiment'sdescriptor to the Launcher. *(2)*Launcher selects a set o nodes according to the received descriptor and deploys the experiment.

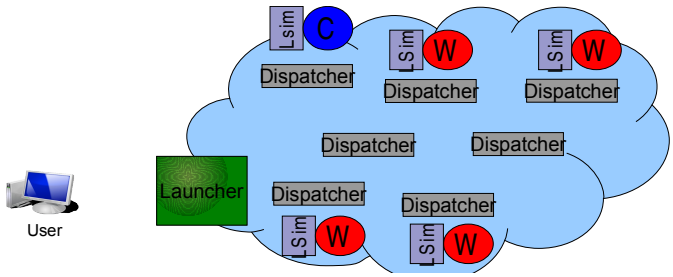

Figure 6. Steps 3 to deploy an experiment using LSim framework. Each selected dispatcher instantiates a coordinator (C) instance or a worker (W) instance of the application of protocol.