VM instantiation and Petri net analysis: two case studies on gLite Grid

Francesco Longo
Università degli Studi di Messina
Messina - 18/05/2011
flongo@unime.it
Virtual Machine on GRID

- Software: Virtual Box 4
- Platform: GRID
- Language: Java, Bash Scripting
Installation and storing

- Virtual Machines automatic creation through (java) graphic interface
- Unattended windows (xp/vista) installation on VMs
- Saving machines and hard disks (.vdi storaging unity) on GRID SEs
Recovery and startup

- Recovery of the list of saved machines and disks from SE
- Job submission to start a virtual machine
- Submission of a specific VM and associated disk to a GRID WN to start it
- Establish a rdesktop connection to the WN to access the machine
- Stop windows session and update of the modified disk on GRID SE
JAVA user interface

- The user interface starts from user home on GRID UI
- The user sets machine and operative system parameters
- He/she also choose to include some program
- The application will automatically make available the new machine with windows installed and configured
The user interface starts from user home on GRID UI

The application searches stored VMs and related disks

The user selects a VM

The application produces the JDL file to submit the job on GRID
JAVA user interface

- The user interface starts from user home on GRID UI
- The application creates an rdesktop connection (on GRID UI) to the WN where the job is running
How it works

- **New Machine**
  - When the user wants to create a new VM, he sets two kinds of field related to the virtual hardware and the windows installation parameters (user name, user password, version XP/VISTA)
  - The Java application makes an unattended installation file with user options and puts it in a new iso file that is created runtime
  - The VM creation and Windows installation is performed on the GRID UI within the user's home directory
  - When installation is over, the machine and its disk are automatically stored on the GRID SE
How it works

- Search VMs
  - The Java application searches on SE for saved machines and shows the results to user
  - The user can select the target machine and submit it
How it works

Submission

- The Java application creates a JDL file and a bash script with the selected VM information. The script also contains a unique file name (based on data and time)
- When the job lands on the WN, it writes on the GRID SE a file with the agreed name. In this file the WN writes its hostname so that the Java application can know on which WN the virtual machine started.
- The WN copies the VM files from the SE
- The VM is registered to the hypervisor
- The vrdp server is launched on the VM
- When the machine is turned off from Windows' user the script copy the updated VM files on the GRID SE
How it works

- Connect
  - The Java application waits that the file containing WN name is written and rescues it
  - The application runs the RDP client connecting to the WN and the user can work on the VM
Running...
Future implementation...

In order to make the service usable to final users, a web application is being implemented
Web application

- Granted access by authentication
- Controlled registration by administrator
- Creation, management and use of the machines
- Virtual machines separated storage for each users
- Ajax mechanisms for maximum interactivity
Overview of Petri Nets

- A Petri Net (PN) comprises a set of places $P$, a set of transitions $T$, and three functions defined on the set of transitions.

- The input, output, and inhibition functions, denoted by $I(.)$, $O(.)$, and $H(.)$ respectively, map transitions on bags of places.

- Places may contain tokens.

- The state of a PN, usually called marking, is defined by the number of tokens contained in each place.

- Transitions represent events in the modeled system.
Overview of Petri Nets

- A Petri Net can be graphically represented by drawing places as circles, transitions as thin bars, and input, output, and inhibition functions as oriented arcs from place to transition or vice-versa.
Overview of WebSPN

- WebSPN is a powerful software tool for modeling and analysis of Non Markovian Stochastic Petri Nets (NMSPN).
- It has been partially implemented using the Java technology which ensure high programmer productivity, security over the network, and multi-threading capabilities;
- it is based on a client-server architecture which ensure scalability and accessibility through the Web;
- it allows to graphically represent the NMSPN under analysis and the desired measures.
Webspn Simulator

• The WebSPN simulator can be used when the analytical solution engine is not able to deal with huge models;

• the cons are a relatively longer response time and a non-exact solution;

• all the high level measures can be obtained with the desired confidence interval;

• the NMSPN discrete events simulator has been implemented in C++;

• the user is able to choose between the analytical and the simulative approach by mean of the GUI.
GridSPN tool

- Both the analytical solution engine and the simulator have been ported to the Grid infrastructure;
- the GUI allows the user to choose if he wants to submit jobs to the Grid;
- the Direct Acyclic Graph Job type has been used for the execution of the simulator on the Grid;
- The Grid solution for simulation is used for transitory measures (e.g. Reliability Function).
Architecture of the Grid Simulator
Architecture of the Grid Simulator

- The client sends some data to the server
  - An xml file that contains the model of the Petri Net to be simulated;
  - A text file that contains the measures, written in a specific syntax of WebSPN;
  - Number of runs and number of computing elements into which the simulation runs must be divided;
  - Amplitude of time intervals (Delta).
- The server, which is on a UI, creates a JDL file and scripts for the submission of the job.
Architecture of the Grid Simulator

- The JDL file created by the server application describes a Directed Acyclic Graph (DAG).
- The number of runs is divided into $N$ nodes of the DAG, one for each computing element to be used.
  - Each of these $N$ nodes executes an instance of the C++ WebSPN simulator
- An additional node (merge node) calls a C executable that merges the results of the first $N$ computations.
Architecture of the Grid Simulator: structure of the DAG

n_1 \rightarrow \text{merge node} \rightarrow n_N

\ldots

Output of the simulation
WebSPN simulator on Grid: an example of execution

This is the main window of the WebSPN GUI. We click the “Open” button, to open a Petri Net model.
We select a model to open.
Now the Petri Net is loaded and graphically represented. We click on the “Analysis” button to open the Measures window.
In this window, we write the desired measures in a specific syntax of WebSPN;

- we check the “Simulation” and “Grid” boxes, to indicate that we want to do a simulation on grid;
- we set other needed values, such as delta of simulation, number of machines, confidence level, number of runs of the simulation;
- then we click on Analyze.
We insert a tag for this job submission.

Then we insert the server address and the port to which the server is listening.
In a few seconds, all data is sent to the server, the job is submitted and the output of the submission is put on a text field in the Measures window. Then we can click on the “Grid Operation” button to make other operations on the job (check status, retrieve output, etc.)
In the “Grid Operation” window, we can monitor all jobs. All data of the job is stored on the server, and it is loaded on the client GUI when this window is open. By clicking on a model name on the left, we can see a list of existent jobs on the right.
By selecting the desired job and clicking the “Status Job” button, we tell the server to ask for the status of our DAG job, and return back the output.
Every time we click on the “Status Job” button, we can see the current state of the DAG.
When the job status is Done, we can ask the server for the output. Then, the server will retrieve the output from grid and send it back to the client.
WebSPN analysis with MPI

- An analytical solution is also available
- A parallelized version of the expansion algorithm has been designed;
- the parallelized version has been implemented by using the MPI library;
- the speedup increase almost linearly with the number of parallel processes;
- in the worst case a gain is always assured at least in terms of occupation of memory used on each machine.
Sequential expansion

- Step 0: Initial States
- Step 1: Get an expanded state
- Step 2a: New generated expanded states
- Step 2b: Already generated expanded states

N

EXPANSION PROCESS

DPH_SPN DESCRIPTION
Parallel expansion
Future developments

- A parametrization of Petri Nets is being implemented.
  - We consider a Petri Net that varies with a parameter $k$ (e.g. The number of tokens in a place).
  - This results in a class of Petri Nets. Each of these Petri Nets will be associated to a grid job.