

Enabling OMNeT++-based simulations on Grid Systems

M. Kozlovsky
MTA SZTAKI
Victor Hugo str. 18-22., H-1518,
P.O.Box 63., Budapest, Hungary
+36-1-2796072
m.kozlovsky@sztaki.hu

A. Balasko
MTA SZTAKI
Victor Hugo str. 18-22., H-1518,
P.O.Box 63., Budapest, Hungary
+36-1-2796072
balasko@sztaki.hu

A. Varga
OpenSim Ltd.
Szőlő köz 11
H-1032 Budapest, Hungary
+36 1 3882763
andras@omnetpp.org

ABSTRACT

Simulation involves using a model to produce results, rather than experimenting with a real system. Simulations are typically computationally intensive problems, and lend themselves for execution on large-scale PC clusters or grids. Use of grid infrastructure for discrete event simulation like communications network simulations, or queuing network simulations etc. is not prevalent; making grid technology easily accessible to simulator framework users can change that picture significantly. In this paper we give a detailed overview how the OMNeT++ simulation framework was ported onto a gLite-based Grid infrastructure. The porting of the simulation framework to the grid infrastructure was supported by the GASUC Team of the EGEE III. project. Later on in the paper we show an example grid service which is able to execute queuing network simulations, and assess its performance on the grid.

General Terms

Algorithms, Performance, Design.

Keywords

Grid, OMNeT++ simulation framework, gridification, grid infrastructure

1. INTRODUCTION

Distributed computer environments are using different technologies to coordinate their processing power, job scheduling, and storage consumption. Porting legacy applications onto grid infrastructures is called "gridification". Porting the OMNeT++[1] framework to grid is useful for users running simulations with large-scale Parameter Studies (PS). Speeding up simulations with large parameter studies can be done easily, by launching the applications on large computer farms. Potential users or beneficiary community of the grid-enabled application can be anyone running simulations with large parameter studies (the total execution time should be well over a few hours for a user to feel the need for grid; for example, when one simulation run takes 10 minutes and 60 runs are needed for a study, that's enough incentive to put in that extra effort needed for grid execution).

1.1 OMNeT++

OMNeT++ is a public-source, component-based, modular, discrete event simulation environment. Due its strong GUI support and embeddable simulation kernel it is frequently used for

simulation of communication networks, IT systems, queuing networks and for various business processes. OMNeT++ provides mainly component architecture for models. Components (modules) are programmed in C++, and then assembled into larger compound models using a high-level language (NED). OMNeT++ supports different platforms, such as Linux, various Unix-like systems and Windows (XP and 2000).

1.2 GASUC Team

The Grid Application Support Centre (GASuC) supported by the EGEE III project¹, provides assistance to grid users and application developers during the application gridification process. GASuC helps identify and apply best patterns, practices, tools and infrastructures in order to decrease the porting time and get application code running on production grids. The generic support model of GASuC consists 8 main steps: 1. Contact phase, 2. Pre-selection phase, 3. Analysis phase, 4. Planning phase, 5. Prototyping phase, 6. Testing phase, 7. Execution phase, 8. Dissemination and feedback phase.

1.3 P-GRADE Grid Portal

The P-GRADE Grid Portal (Parallel Grid Run-time and Application Development Environment) [4] is an open source, service-rich, workflow-oriented graphical grid portal environment. It supports workflows composed of sequential jobs, parallel jobs and application services. The P-GRADE Portal hides the complexity of the Grids through its high-level graphical Web interface, which can be used to develop and execute and monitor workflow applications on Grid systems built with Globus, EGEE (LCG or gLite) and ARC middleware technologies.

¹ EGEE-III is an FP7 project funded by the European Commission under contract INFSO-RI-222667.

1.3.1 Workflow concept of the P-GRADE Portal

The P-GRADE Grid Portal environment is using a DAG (directed acyclic graph) based workflow concept (shown in Figure 1.).

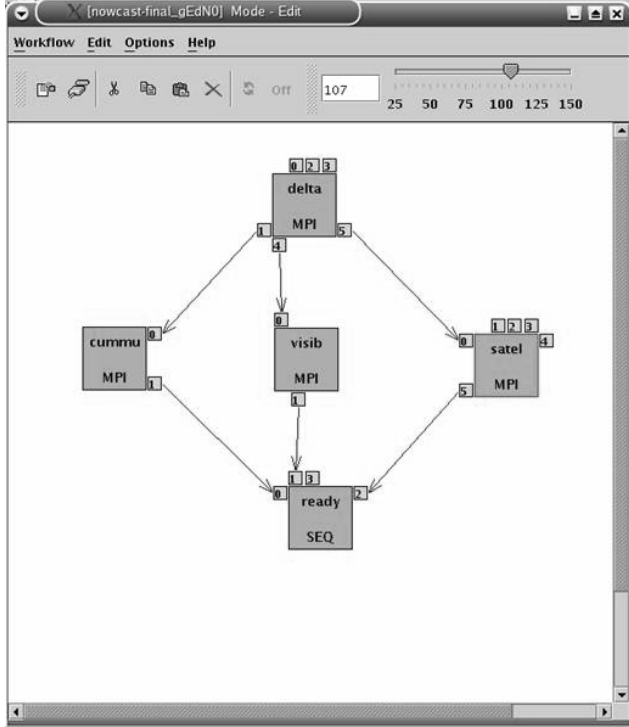


Figure 1. Example workflow in P-GRADE Portal.

In a generic workflow, nodes (shown in Figure 1. as large gray squares) represent jobs, which are basically batch programs to be executed on a computing element. Ports (shown here as small squares around the large ones) represent input/output files the jobs receiving or producing. Arcs between ports represent file transfer operations. The basic semantics of the DAG-based workflow is that a job can be executed if and only if all of its input files are available.

1.3.2 Parameter Study support in P-GRADE Portal

According to our experiences, user communities have great claim to be able to run programs parallel with different input files. P-GRADE Portal supports this kind of parallelization called Parameter Study, or Parameter Sweep on a high level.

The original “job” idea has extended by two special jobs, called Generator and Collector to make flexible the development of Parameter Study type workflows in P-GRADE Portal. The Generator job is used to generate the input files for all parallel jobs automatically (called Automatic Parameter Input Generator), or by a user-uploaded application (called Normal Generator), while the Collector will run after all parallel execution and collects all parallel outputs as input files [5].

All jobs between these special ones, by the side of Arcs will run as much instances as input files are generated by the Generator. This is the main idea of Parameter Study in P-Grade Portal. During the gridification, we have exploited these possibilities of P-GRADE Portal.

2. Gridification process

Main steps of the gridification process are shown on Figure 2. The compilation of the source code on the UI machine assures that the application will work on all the gLite-based Computing Elements in the grid infrastructure. Standalone, single run tests showed that we need to have the compiled binary code, the predefined NED and INI files as inputs for the porting of the simulation into a grid workflow.

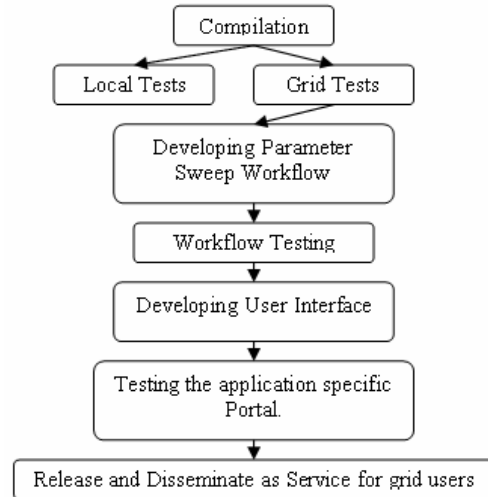


Figure 2. Gridification steps of the simulator.

We have also investigated the time and storage consumption and running environment requirements (e.g. shared libraries: libstdc++) of the simulator. For the first grid tests (single job tests), we have written a JDL (Job Description Language) file and created the workflow graph of the simulation in the P-GRADE Portal environment. The parallel job submission into the grid environment needs to have parameter assignment of the generated parameters. P-GRADE Portal’s PS workflow components and OMNeT++’s parameter study solution (a detailed description about the PS handling in OMNeT++ follows in Section 3.) was integrated together to create a grid-aware OMNeT++ simulator environment and realize a complex grid workflow as a proof of concept. Later on we have developed the web-based grid user interface for OMNeT++ simulator framework with the help of the Application Specific Module of the P-GRADE portal. On this web interface, end-users can upload their network topology and configuration files for queuing network simulations, and they can submit the simulation into the grid environment with large parameter fields.

3. Parameter Study handling in OMNeT++

3.1 Overview

In a simulation study, we create a simulation model, then perform experiments on it in order to learn about its behavior under various conditions. OMNeT++ emphasizes separation of concerns, which in this case means that the model and its input parameters are in physically separate files. The simulation model is represented by C++ code (and the libraries/executables built from them) and NED files describing the topology and fixed parameters. Input parameters for the experiments are in INI files, which also contain configuration settings for the simulator.

An INI file (the default name is `omnetpp.ini`) may contain several named configurations. Every named configuration contains parameter assignments, each of which typically sets more than one model parameters, using wildcards (i.e. `**.tcp.mss=1024` sets the MSS parameter for all TCP instances in the model). Named configurations may also contain simulator settings (stopping condition, seeds, etc), and they can also inherit from each other which makes it possible to factor out common parts. For every simulation run, a named configuration needs to be specified where all parameters and settings will be taken from.

A named configuration may also contain iterations and repetitions, to support parameter studies. Thus, a configuration may describe several simulation runs. They are identified with run numbers, in the range $0..n-1$. Each run may be executed independent of the others. To perform a specific run, the user has to tell the configuration name and the run number to the simulation runtime, typically via command-line arguments. The simulation runtime may also be asked about the number of runs a particular configuration generates, or to enumerate the settings for each run.

By default, result files will be created in the `result/` subdirectory of the simulation. Each run generates separate files, by default named after the configuration and the run number. Result files are self-describing, and contain enough metadata so that later it can be identified how they were created.

Every run also gets a globally unique identifier, called a *run ID*, concatenated from the configuration name and run number, date/time, process ID, etc.² Run IDs are also recorded into the files, and can be used (for example) to relate different files created by the same run.

3.2 Iterations

The value part of a parameter assignment in the INI file may contain iterations. Syntax of an iteration is `${valuelist}` or `$(name=valuelist)`, where *valuelist* is a comma-separated list of values, and *value* is some arbitrary text (typically a numeric constant). There is also a shorthand for generating numeric sequences, with the syntax `start..end` or `start..end step increment`. Each value generates a separate simulation run, where the value gets textually substituted into the parameter assignment before the parameter value gets interpreted.³ If there is more than one iteration in a named configuration, they are combined with Cartesian product, i.e. they essentially form nested loops. It is also possible to specify a constraint expression, to take a subset of the Cartesian product. In the constraint expression, one can refer the current iteration values with the syntax `$(name)` or `$0`, `$1`, etc. This iteration variable syntax also makes it possible to refer to an iteration value in parameter assignments other than which defined it.

² Run IDs are only globally unique within reasonable limits, because we held it more important that they are both short and readable.

³ This macro-style implementation was chosen because in OMNeT++ it would be impractical to iterate over individual parameters directly, given that each parameter assignment in an INI file typically assigns several parameters together, using wildcard expressions. Macros also provide more flexibility, much in the same way as C macros can do things not possible with C functions.

A special parallel iteration syntax makes it also possible that an iteration does not form a Cartesian product with another iteration, but gets advanced in lockstep with it. Given the iterations `$(N=10,20,50,100)` and `$(D="low","medium","high","veryhigh" ! N)`, four simulation runs will be generated, with *N* and *D* being `(10,"low")`, `(20,"medium")`, `(50,"high")`, and `(100,"veryhigh")`.

There are also predefined variables: `$(configname)`, `$(runnumber)`, `$(runid)`, `$(datetime)`, `$(processid)`, etc. They can be useful for generating file names, for example.

3.3 Repeating runs with different seeds

It is directly supported to perform several runs with the same parameters but different random number seeds. There are two configuration options related to this: `repeat` and `seed-set`. The first one specifies how many times a run needs to be repeated. For example, `repeat=10` causes every combination of iteration variables to be repeated 10 times, with the `$(repetition)` predefined variable being the loop counter; thus, `repeat=10` is essentially equivalent to adding `$(repetition=0..9)` to the ini file. The `$(repetition)` loop always becomes the innermost loop.

The `seed-set` configuration key affects seed selection. Every simulation uses one or more random number generators (as configured by the `num-rngs` option), for which the simulation kernel can automatically generate seeds. The first simulation run may use one set of seeds (seed set 0), the second run may use a second set (seed set 1), and so on. All automatic seeds generate random number sequences that are far apart in the RNG's cycle, so they will never overlap during simulations. (OMNeT++ uses Mersenne Twister, which has a cycle length of 2^{19937} .)

3.4 How to run

To run a parameter study expressed with iterations as described above, one first needs to find out how many runs the iterations unroll to. This is achieved by running the simulation executable with the `-x <configname>` command-line option, which in turn prints out the *n* number of runs in the given named configuration. Individual runs can then be executed by running the simulation program with the `-c <configname> -r <runnumber>` options, where *runnumber* is an integer in the $0..n-1$ range. The fact that each run can be executed independently enables them to be submitted to grids as well. Output files will be named after the configuration and the run number (although this is configurable as well).

3.5 Experiment-measurement-replication

OMNeT++ uses the concepts *experiment*, *measurement* and *replication* to organize simulation results generated by batch executions or several batches of executions. During a simulation study, a person prepares several experiments. The purpose of an experiment is to find out the answer to questions like "how does the number of nodes affect response times in the network?" By default, an experiment is represented by a named configuration in the ini file, but it is also possible to group several configurations into a single experiment. For each experiment, several measurements are performed on the simulation model, and each measurement runs the simulation model with a different parameter settings. To eliminate the bias introduced by the particular random number stream used for the simulation, several replications of every measurement are run with different random number seeds. Measurements map to iteration variables in the ini file, and replications map to the `repeat` option.

In order to make result analysis tools aware of the above concepts, OMNeT++ saves an experiment label, measurement label and

replication label into result files. The labels can be configured in the ini file, but the suitable defaults are "\$*{configname}*", "*{iterationvars}*" and the "#*{repetition}*, *seed-set=<seedset>*" text.

4. Grid workflow structure

The grid aware queuing model workflow was developed using the P-GRADE Portal's Workflow Editor. A simple queuing model diagram (Tandem Queue) is depicted in Figure 3.

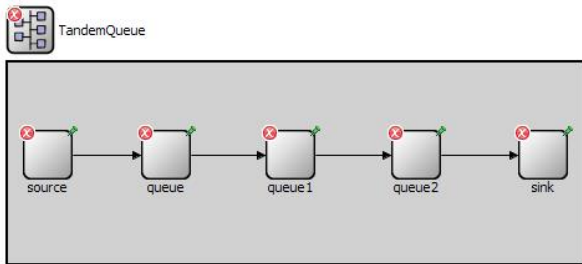


Figure 3. Simple queuing model diagram from OMNeT++

To enable the execution of the model as a parameter study application, we have used the "Automatic Parameter Input Generator", and the "Collector" elements of the P-GRADE Portal.

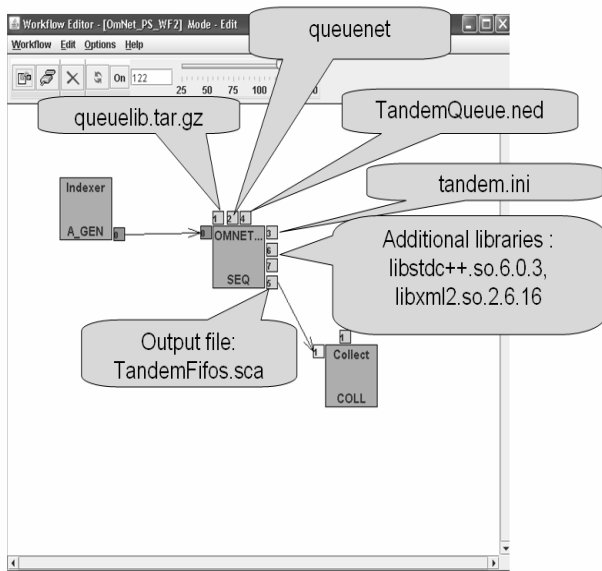


Figure 4. Developed workflow diagram with PS support from P-GRADE Portal's Workflow Editor

The upper left box (shown in Figure 4.) is the Automatic Parameter Input Generator job, which automatically generates index numbers or interval of indexes, and separates the runs by providing instructions which run should be executed.

The second box in the middle contains the application in the form of a pre-compiled binary, the input parameter and topology files (INI and NED files) and some additional libraries. As an output, it can produce the vector or result files of the model, which are collected together by the third "Collector" box (lower right). At the end of the workflow, the Collector job automatically

compresses the results and enable users to download the compressed file for further user analysis.

5. User interface development for grid-aware OMNeT++ models

We have created an easy-to-use service on the web for queuing model simulations. On this web interface end users, who are not grid professionals, are able to upload their configuration and topology files, setup and submit their model into the grid infrastructure. The results produced by the submitted simulations are downloadable by normal web browser when they are finished. The developed web-based (portlet) user interface has three main views:

- The "OMNeT++ Measurement Manager" portlet. This is where the end user can choose among the available OMNeT++-based model workflows (shown in Figure 5.).



Figure 5. OMNeT++ Measurement Manager portlet

- The "Upload inputs" portlet. This is where the end user can upload the queuing topology (NED) and configuration parameter (omnetpp.ini) files, and select the indexes or interval of indexes to run (shown in Figure 6.).

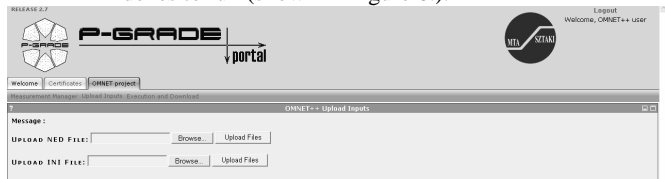


Figure 6. Upload inputs portlet

- The „Execution and download" portlet. This is where the end user can execute the model by a single button (which means basically a complex workflow submission procedure into the grid infrastructure) with its parameter field. (shown in Figure 7.).

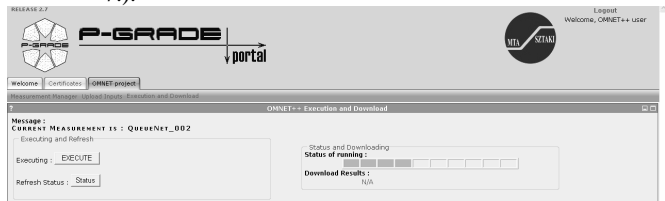


Figure 7. Execution and download" portlet

When simulation execution has finished successfully, the result files of the simulation can be downloaded with the "Download" button.

6. Performance

The EGEE and SEE-GRID-SCI grids were used for submission of the developed workflow model. We have used several times a simple queuing model to test our porting concept. In our largest parameter study demo, the simulation time of the used queuing model was only 1 minutes, however the parameter field of the model generated 5000 “runs”, which would last about 80 hours to execute on a single average computer . We have clustered the runs with the help of the “Automatic Parameter Input Generator“ into smaller (100-run) chunks, and submitted the model into the grid infrastructure.

The grid broker allocated 50 computers for these chunks from the grid as it was requested. The results of the parallel grid submissions were available after 3 hours, instead of 3.5 days. Each running generated 25-225 kBytes of data (the compressed sca files), size of the final output was about 6MB.

7. Conclusion

We have successfully integrated the P-GRADE Grid Portal environment with the OMNeT++ simulation framework to enable additional, already existing, large-scale grid resources to the simulation user community. We have used the parameter study features from both (P-GRADE Portal and OMNeT++) environments to create a generic workflow solution for OMNeT++-based simulation models. We have shown that large-scale grid infrastructure can provide significant performance increase for OMNeT++-based simulations. As a proof of concept, we have created from the developed queuing workflow a web-based service, where a configurable queuing model can be

submitted by the end-user into the grid infrastructure without any grid knowledge. If the developed workflow and the service solution can grow beyond the concept phase, they have the potential to become a useful tool for developers and end-users of simulations.

8. REFERENCES

- [1] OMNeT++ Home Page. <http://www.omnetpp.org> [accessed on November9, 2008]
- [2] Varga, A. 2001. The OMNeT++ Discrete Event Simulation System. In the Proceedings of the European Simulation Multiconference (ESM2001. June 6-9, 2001. Prague, Czech Republic).
- [3] Varga, A, Hornig, R. 2008. An Overview of the OMNeT++ Simulation Environment. In the Proceedings of First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008. March 3-7, 2008. Marseille, France).
- [4] Cs. Nemeth, G. Dozsa, R. Lovas, P. Kacsuk: The P-GRADE Grid Portal. ICCSA 2004: International Conference Assisi, Italy, LNCS 3044, pp. 10-19
- [5] P. Kacsuk, Z. Farkas; G. Herman. Workflow-level parameter study support for production grids. Computational science and its application - ICCSA 2007. International conference. Part III. Kuala Lumpur, 2007. (Lecture notes in computer science 4707.); 2007.: Berlin, ISBN: 978-3-540-74482-5, pp. 872-885.