A Powerful Tool-Chain for Setup, Distributed Processing, Analysis and Debugging of OMNeT++ Simulations*

Thomas Dreibholz
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
dreibh@iem.uni-due.de

Erwin P. Rathgeb
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
rathgeb@iem.uni-due.de

ABSTRACT

In this paper, we introduce our Open Source tool-chain providing the parametrization, distributed execution, results post-processing and debugging for OMNeT++-based simulations. While the initial motivation of these tools has been the support of our simulation model of the Reliable Server Pooling (RSerPool) framework, it has been particularly designed with model-independence in mind. That is, it can be easily adapted to other simulation models and therefore may be useful for other users of OMNeT++-based simulation models as well.

Keywords: Simulation Model, Parametrization, Simulation Run Distribution, Plotting, Analysis

1. INTRODUCTION

Reliable Server Pooling (RSerPool) is the IETF's upcoming standard for an application-independent, light-weight framework for the management of server pools [8,11] and sessions [13]. It has been designed in order to ensure the availability of critical services. An important sub-topic of RSerPool is server selection within pools. As proof of concept for RSerPool, we have started the development of the RSPLIB [3] Open Source prototype implementation [2] in 2001. However, in order to analyse, evaluate and optimize the RSerPool approach in detail, a prototype has been insufficient and a simulation model was needed.

After comparing OMNeT++ [33], NS2 [21] and the commercial OPNET [22] frameworks, we have finally chosen OMNeT++ as the foundation of our RSerPool simulation model RSPSIM [9]: in comparison to NS2, the object-oriented structure of OMNeT++ is clearer and easier to understand – and the NED and message object generation tools save a lot of time. The commercial OPNET package has been found too complicated and furthermore it is also extremely expensive.

In order to efficiently perform simulations using our RSPSIM model, we have also developed a model-independent, flexible and powerful tool-chain for the setup, parallel run execution, results aggregation, data analysis and debugging – completely based on Open Source software. The goal of this paper is to present our tool-chain – which is illustrated in figure 1 – in detail to show how the challenges of simulation execution can be solved efficiently. Now, we apply this tool-chain not only for our RSerPool model RSPSIM, but also for two other projects. Due to its independence of a specific model, it may also be useful for many more users of OMNeT++. Our tools have been released as Open Source under GPLv3 license and are freely downloadable from our web site[1].

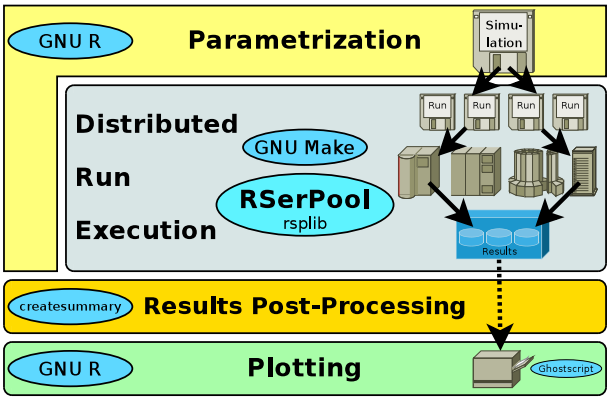

Figure 1: An Overview of Our Tool-Chain

2. THE RSERPOOL ARCHITECTURE

Before describing our tool-chain itself, we first have to introduce RSerPool, since it will also be used for the simulation run distribution later in section 5. Figure 2 illustrates the RSerPool architecture [3,20] which contains three types of components: servers of a pool are called *pool elements* (PE), a client is denoted as *pool user* (PU). The *handlespace* – which is the set of all pools – is managed by redundant *pool registrars* (PR). Within the handlespace, each pool is identified by a unique *pool handle* (PH).

2.1 Components and Protocols

PRs of an *operation scope* synchronize their view of the handlespace using the Endpoint haNdlespace Redundancy

*Parts of this work have been funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).

OMNeT++ 2008 March 3, 2008, Marseille, France

[1] `http://www.iem.uni-due.de/~dreibh/omnetpp/`.

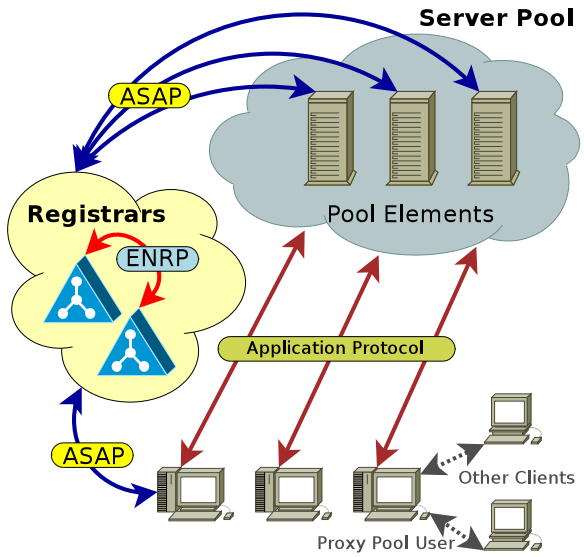

Figure 2: The RSerPool Architecture

Protocol (ENRP [35]), transported via SCTP [19]. An operation scope is restricted to a single administrative domain (e.g. an organization or department), which keeps the management complexity small [8,11]. Being "light-weight" is the fundamental property of RSerPool [3]: it must also be usable on low-performance devices (e.g. routers or embedded systems). Therefore, the duty of RSerPool is the management of pools and sessions only, but it allows for a very efficient realization [11]. Nevertheless, PEs may be distributed globally, so that their service can survive localized disasters [12] (e.g. an earthquake or flooding). PRs can announce themselves to PEs, PUs and other PRs via UDP-based multicast messages. This functionality allows for the automatic configuration of all components.

PEs choose an arbitrary PR of the operation scope to register into a pool by using the Aggregate Server Access Protocol (ASAP [29]), again transported via SCTP. Within its pool, a PE is characterized by its PE ID, which is a randomly chosen 32-bit number. Upon registration at a PR, the chosen PR becomes the Home-PR (PR-H) of the newly registered PE. A PR-H is responsible for monitoring its PEs' availability by keep-alive messages (to be acknowledged by the PE within a given timeout) and propagates the information about its PEs to the other PRs of the operation scope via ENRP updates. PEs re-register regularly (in an interval denoted as *registration lifetime*) and for information updates.

In order to access the service of a pool given by its PH, a PU requests a PE selection from an arbitrary PR of the operation scope, using ASAP transported via SCTP. The PR selects the requested list of PE identities by applying a pool-specific selection rule, called *pool policy*. A basic set of adaptive and non-adaptive pool policies is defined in [30]. For this paper, only Least Used (LU) is relevant: LU selects the least-used PE, according to up-to-date application-specific load information. Round robin selection is applied among multiple least-loaded PEs [8]. Details on other policies can be found in [3].

The PU writes the list of PE identities selected by the PR into its local cache (denoted as *PU-side cache*). From this cache, the PU selects – again using the pool's policy – one element to contact for the desired service. The PU-side cache constitutes a local, temporary and partial copy of the handlespace. Its contents expire after a certain timeout, denoted as *stale cache value*. In many cases, the stale cache value is simply 0s, i.e. the cache is used for a single handle resolution only [9].

2.2 Application Scenarios

Although the main motivation to define RSerPool has been the availability of SS7 (Signalling System No. 7 [18]) services over IP networks, it is intended to be a generic framework. There has already been some research on the performance of RSerPool usage for applications like SCTP-based mobility [6,7], VoIP with SIP [1], web server pools [3], IP Flow Information Export (IPFIX) [5,24], real-time distributed computing [3,4,9,10,14,36–39] and battlefield networks [31]. A further application is the scripting service, which we will introduce later in section 5.

3. THE SIMULATION MODEL

An overview of our RSPSIM simulation model [3] is presented in figure 3: each setup has one *Controller* module which takes care of simulation startup, shutdown and collection of global statistics. The actual RSerPool network setup is provided by an array of *LAN* modules. Each LAN consists of arrays of *Registrar*, *PoolElement* and *PoolUser* modules – all interconnected by a switch. The PU and PE modules realize a generic application model to evaluate the load balancing and failover features of RSerPool. Details on this application model can be found in [3,9].

The implementation of the switch is actually realized by a *TransportNode* module, which is also a sub-module of the PR, PE and PU modules. A TransportNode is an abstraction of the Network and Transport Layers. It provides the forwarding of data packets using addresses and port numbers, actually realized using OMNeT++'s *cTopology* class and shortest-path algorithm. This is already sufficient for our research on RSerPool functionality. However, in the future, the simple TransportNode could be easily replaced by a full-featured SCTP/IP stack like [26].

Handlespace management – which means the storage and maintenance of a handlespace – is an important task of RSerPool. While the naïve solution would simply use a list of pools and store each pool as a list of PE identities, this solution would not scale to large handlespaces (i.e. hundreds or thousands of PUs and PEs). In order to achieve reasonable execution times, a sophisticated solution based on red-black trees has been developed. Details can be found in [8,11]. Since the task of handlespace management is identical for simulation model and prototype implementation, we use our approach for both. But since the prototype uses ANSI C as implementation language, it also had to be used for the handlespace management. Due to `opp_makemake`'s lack of support for `.c` files, we have applied a very simple trick: for each `.c` file, there is also a `.cc` file which simply contains an `#include` statement for the corresponding C code file.

4. THE PARAMETRIZATION

The latest version of the RSPSIM model's network contains almost 120 parameters. So, manually writing `.ini` files for

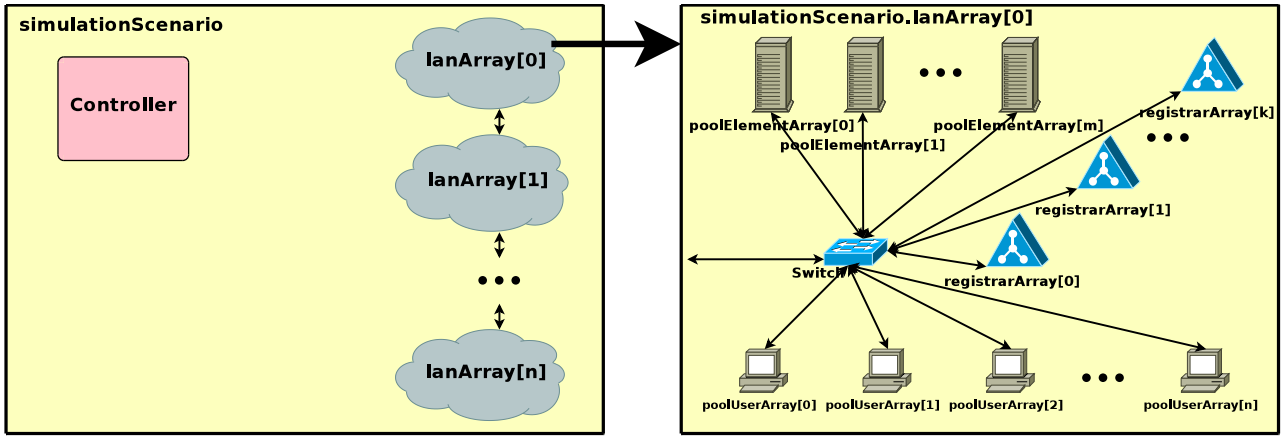

Figure 3: An Overview of the Simulation Model

the model is quite inefficient. In order to cope with this challenge, we have developed a simulation execution script in GNU R [25]. The GNU R package contains a powerful script language and a huge number of extensions for statistical analysis and plotting.

In order to explain the first step of model-independent simulation processing – the parametrization – it is useful to introduce some formal definitions: let a simulation model have n *parameters* $p_1, \ldots, p_n$, with $\hat{P}_1, \ldots, \hat{P}_n$ the corresponding *parameter spaces*. That is, $p_i \in \hat{P}_i$ for all $i \in \{1, \ldots, n\}$. Then, the *model parameter space* is $\hat{P} = \hat{P}_1 \times \hat{P}_2 \times \ldots \times \hat{P}_n$. Using this definition, a *simulation* $S \subset \hat{P}$ simply contains all parameter combinations $s \in S$ for which a *run* has to be performed. We assume for simplicity reasons that a run number corresponding to a certain random number generator seed is simply another input parameter. The simulation binary itself constitutes a *simulation function* $f : S \to R$, which maps a run $s_j \in S$ to a result $f(s_j) \in R$ (scalars and vectors; we omit a formal definition here). We further assume that for the same setting of s, always the same output is generated (or differences do not falsify the results[2]).

Clearly, the first step to parametrize a simulation model is to define the simulation S. Listing 1 presents a simple example from RSPSIM: `simulationConfigurations` is a list containing sub-lists. Each sub-list includes the parameter name (as first item) and all values to be used.

In order to generate input for the model, `.ini` files have to be written. For efficiency reasons, our simulation script has to meet two goals:

Extensibility It must be possible to add more values for some parameters, without having to re-process already performed runs.

Parallelization There must be support for executing different runs simultaneously, using our run distribution approach described in section 5.

To fulfil these requirements, our simulation script first creates a separate *run directory* for each run $s \in S$. The name of the directory is computed using the SHA1 hash [15] over s, which avoids overly long names. For each run s, a separate `.ini` file is generated. It also uses its own scalar and vector files. Furthermore, a Makefile for GNU MAKE [17] is written for the whole simulation S. Each $s \in S$ leads to an entry performing the following tasks:

- Removal of old vector, scalar and log files.
- Execution of the simulation model binary using the corresponding `.ini` file for s. It will write a log file as well as – probably – scalar and vector files.
- Compression of output files using BZIP2 [27]. Since the output files are simply ASCII text, this can achieve a huge space reduction.
- Creation of a time-stamp file after successfully processing all former steps.

A re-run of the simulation script will update existing time-stamp files by default. That is, already executed runs will not be re-processed again – since their result would not change (due to our assumption for f above). If the simulation function f changes, the update step can be skipped and the runs will be executed again. Since run directories are kept until manually deleted, we also get caching behaviour: if the simulation is modified from S to $S' \subset P$, $S' \neq S$, it is only necessary to process the new runs $s \in S' \setminus S$. Note, that the runs $\overline{s} \in S \setminus S'$ are still kept. They may be reused again after further modification of the simulation.

In order to make the simulation script reusable, it is separated into a model-independent part (for generating all $s \in S$, creating directories and files as well as writing the contents of the Makefile) and a model-specific part. The model-specific functionality consists of writing the parameter section of the `.ini` file.

5. DISTRIBUTING SIMULATION RUNS

5.1 Overview

The execution of a simulation S simply consists of processing the generated Makefile by GNU MAKE. This is realized as last step of the simulation script. In particular, the simulation script also finds out the number of CPUs/cores[3] and

[2]For example, the RSPSIM model also writes the actual run execution time as a scalar, in order to allow for profiling.

[3]By using the CPU information from `/proc/cpuinfo`.

Listing 1 An Example Simulation Configuration

```
simulationConfigurations <- list(
   # ====== Variable Settings ==========================================
   list("targetSystemUtilization", 0.80),
   list("puToPERatio", 1, 2, 3, 4, 5, 7, 10, 15, 20),
   ...

   # ====== Pool Element Settings ======================================
   list("calcAppPoolElementServiceCapacityVariable", 1000000),
   list("calcAppPoolElementSelectionPolicy", "LeastUsed", "Random", "RoundRobin"),
   ...

   # ====== Pool User Settings =========================================
   list("calcAppPoolUserServiceJobSizeVariable", 1e6, 1e7, 1e8),
   ...
)
```

lets GNU MAKE execute the appropriate number of runs in parallel[4]. That is, a dual-core machine should perform two runs simultaneously. However, this approach is still limited to a single PC only.

In other to allow for parallel simulation processing in our networking lab and on some spare PCs, we have first considered AKAROA. However, the configuration in our quite heterogeneous network (different Linux versions, different subnets, downtime when PCs are used for student exercises or projects, etc.) has been challenging and a "light-weight" approach for simulation distribution has been desired. But RSerPool itself is a light-weight framework for request distribution in server pools. Furthermore, we have developed our prototype implementation RSPLIB and it is even installed on our PCs. So, it has been quite straight-forward to utilize RSerPool to do this job!

5.2 Using the Scripting Service

The RSPLIB [2] package already contains the "scripting service" (SS) as application demo. For this service, a PU can establish a session with a pool and upload a TAR/GZIP-packed archive to a PE. The selected PE unpacks the archive into a temporary directory and executes a script included in the archive. This script can write an output archive, which is finally downloaded to the PU. The scripting pool can use the Least Used policy. Each PE can handle up to SSMaxThreads sessions simultaneously [36]; a PE's load value is set according to its actual number of sessions.

Using the scripting service for our simulation processing is easy: instead of invoking the simulation model binary in the Makefile itself, the script `ssdistribute` is called. This script simply packs the `.ini` file and a script called `ssrun` into a TAR/GZIP archive and provides it to the scripting service PU. The PU will distribute this archive to a PE in the simulation computation pool and the PE will execute `ssrun`. `ssrun` will actually call the simulation model binary, collect scalar, vector and log files and put them together into an archive. This archive is downloaded by the PU and stored in the corresponding simulation directory. If a PE rejects a session (since already serving SSMaxThreads sessions), or if it goes out of service (e.g. the PC is turned off), the session is simply restarted from scratch ("abort and restart" principle [13]) after a short delay (e.g. 5s). This delay avoids overloading the network with reject-and-retry floods [37] when there are too few PEs available.

Using the scripting service of RSerPool is a quite simple way for the simulation distribution: `ssdistribute` and `ssrun` each consist of about 50 lines of shell code. Setting up a simulation computation pool now gets rather easy: figure 4 illustrates our group's setup consisting of 27 cores provided by 15 PCs. On each computation PC, it is only necessary to start a PE with SSMaxThreads set to the system's number of CPUs/cores. Each PE will automatically find a PR, therefore no configuration is required. Of course, it is possible to dynamically add or remove PEs. That is, when the PCs of our student lab are required for other tasks, the scripting PE may be stopped. On the PU side, GNU MAKE has to be called with an appropriate number of simultaneous processes. Then, there will be up to the given number of parallel simulation run sessions.

In fact, it would only be necessary to install the scripting PE on the computation PCs. However, for our RSPSIM model, we also provide the simulation model binary on the PCs themselves – instead of providing it in the TAR/GZIP archive sent by the PU. This saves some bandwidth and furthermore allows mixed pools of x86 and x86_64 machines and different Linux distributions.

5.3 Work in Progress

At the moment, we are testing further improvements of the distribution service: first, we would like to use transparent application checkpointing [23] to regularly create snapshots of the running OMNET++-based simulation model. So, instead of applying "abort and restart" upon PE failure, it would be possible to resume a run from the latest checkpoint. Another point of improvement is security: the scripting service executes arbitrary scripts sent from a remote instance. This is fine for our lab PCs, but prevents application on other systems. So, instead of running such scripts as regular processes, we would like to use a XEN-based virtual machine. Our long-term objective is to run a virtual machine-based scripting service on user PCs, in order to let them safely provide their capacity to simulation processing (or some other useful tasks) when idle.

6. THE RESULTS ANALYSIS

After processing of a simulation S, the simulation directory contains a BZIP2-compressed scalar and/or vector file in the sub-directory of each run $s \in S$. Clearly, the results

[4]Using the parameter `-j [jobs]`.

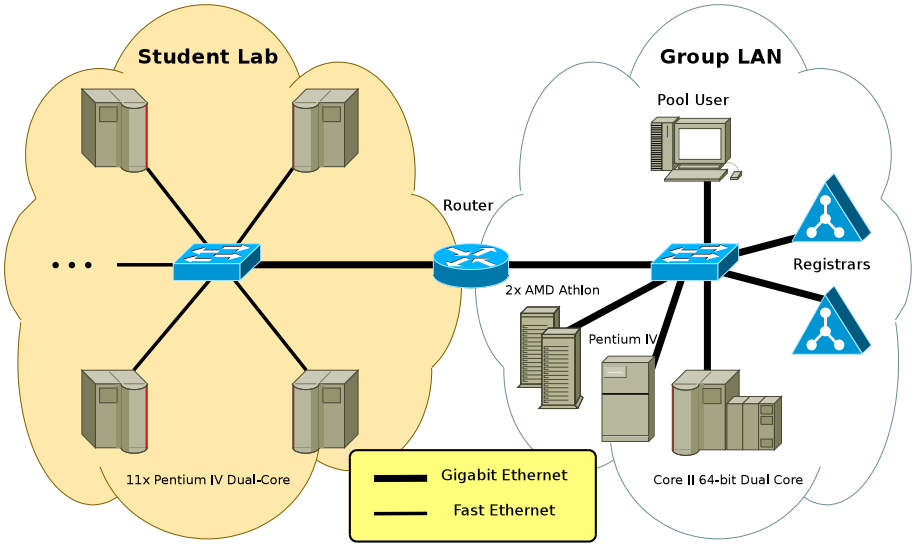

Figure 4: Our Scripting Service Setup for Simulation Run Distribution

Listing 2 An Example GNU R Data File

```
1                Size  Interval  ID    System  Speed
2  0001             1       100  Test  Alpha   39.21
3  0002            20       150  Test  Beta    48.20
4  0003            20       152  Test  Beta    96.03
5  0004            20       155  Test  Beta    12.62
6  0005            50       140  Test  Alpha  139.23
7  0006            75       180  Test  Beta    45.34
8  0007            80       120  Test  Alpha   73.28
9  0008            90       145  Test  Alpha   59.29
10  ...           ...       ...  ...   ...       ...
```

from these files have to be collected and summarized in order to perform an analysis. For the RSPSIM model, we mainly use scalar files, therefore we omit vector file handling in this description. Since we have already used GNU R for our simulation script, it has been straight-forward to also apply this program for the post-processing of the results. However, the mechanisms we describe in the following could be easily adapted to other tools – e.g. GNU OCTAVE [16] and GNU PLOT [34] – as well.

6.1 The Summarization Tool

As first part of the scalar post-processing, all scalar files are read (with BZIP2-decompression on the fly) and the information is stored in memory. Since this task can require a lot of memory, it is realized by a C++-written program called `createsummary`. In order to simplify the post-processing, the run configuration s and the corresponding directory name are remembered at Makefile generation (see section 4).

When all results are in memory, it is easily possible to write them into data files for GNU R. Such a data file is simply a text file containing the column names in the first line. Each following line contains the data, with line number and an entry for each column (all separated by spaces). An example data file is shown in listing 2; it contains the parameters *Size*, *Interval*, *ID* and *System* as well as the scalar *Speed*. Since the RSPSIM model contains about 120 parameters, there would be the same number of columns. However, most parameters are actually fixed for a realistic simulation S. Therefore, such columns are simply not written unless explicitly requested (e.g. if needed for post-processing later). Furthermore, the resulting data files will be BZIP2-compressed on the fly, in order to reduce storage space.

6.2 Plotting

The final step of results post-processing is the graphical representation. For each scalar, there is a BZIP2-compressed data file containing the scalar values as well as the parameter settings used to obtain a corresponding value. Since we have already used GNU R [25] for the parametrization, it is quite straight-forward to also use it for plotting – GNU R also provides powerful graphics functions. In particular, it allows for a very fine-granular control of the output plots to adapt the presentation to special requirements (e.g. labels, grids, colours, line styles, etc.). However, it would also be possible to apply other tools like GNU OCTAVE and GNU PLOT for plotting in a similar way. The requirements to our plot function are as follows:

1. There must be support for multiple lines per plot (Z-axis). Furthermore, lines should be sub-dividable by further parameters (V-axis, W-axis).
2. It must be possible to compute and display confidence intervals.
3. The output should optionally be black and white, grey scale our colour.
4. All plots should be stored in PDF files, for inclusion into pdfLaTeX documents.

The first requirement is achieved by appropriately subsetting the obtained results table. Figure 5 presents an example taken from [3]: on the X-axis, the number ratio between PUs and PEs is varied (PU:PE ratio r), the Y-axis shows the resulting system utilization (which is the value of a scalar).

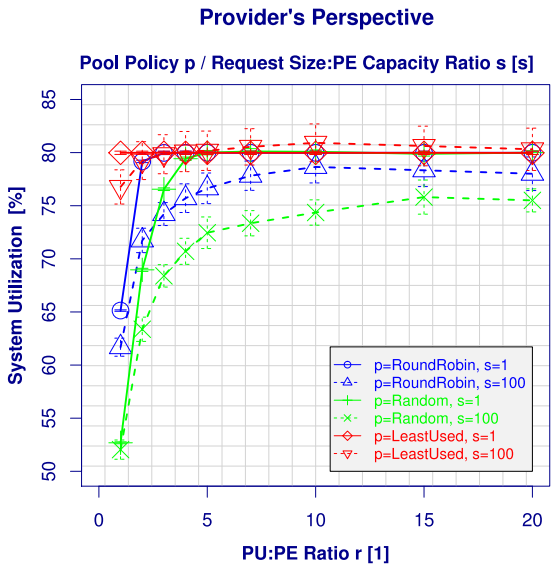

Figure 5: An Example Results Plot

Each X-axis point r is sub-divided by the pool policy p (Z-axis) and furthermore by the request size:PE capacity ratio s[5] (V-axis). For each parameter combination, there have been 24 runs with different seeds. That is, for each X-axis/Z-axis/V-axis parameter combination, there are 24 utilization values in the data table. Our plotter script simply takes these 24 values and computes the average value as well as the 95%-confidence interval. The confidence interval is displayed by thin lines, the average value is used for the actual curves. The colour of each curve (or shade on a black and white printout) is given by the value of the Z-axis (here: pool policy p; colours are automatically chosen for high contrast), the line style (solid, dotted, etc.) by the value of the V-axis (here: request size:PE capacity ratio). That is, we try to present the results as descriptively as possible.

In order to speed up the definition of plots to be created, our script allows for the definition of templates for the mapping of table columns to axes. These templates are used to write the actual plot definitions. Listing 3 provides an example from [3] for two plots: system utilization (this plot is shown in figure 5) and request handling speed. The plot configuration in `plotConfigurations` simply consists of a list of plot definitions; the first line of each definition provides the simulation directory (i.e. where to find the results data) and the PDF output file name (created using the directory name). In the second line, the plot title, optional ranges for X-axis and Y-axis (NA denotes automatic choice) and the legend position are provided. The following definitions provide the template names for the values of the axes (X, Y, Z, V, W).

The templates are defined in `rspsim5PlotVariables`. A template does not only correspond to a certain data table column, it can furthermore also apply data modification. For example, we plot the system utilization in the first plot – which is provided as values from 0.0 to 1.0 in the data file. For readability reasons, we have configured the template (`controller.SystemAverageUtilization`) to multiply it by 100 to obtain a value in %. A template furthermore defines the axis label and the files(s) from which an axis vector is actually read. Using our template-based definitions, plots can be created very easily.

GNU R is already capable of writing its plots into PDF files. However, there are two limitations: the output can use the latin encoding only, i.e. without special characters of non-English languages. Furthermore, no fonts are embedded into the PDF file. That is, when included into pdfLaTeX, the resulting file will contain non-embedded fonts. Such files are e.g. not allowed for the camera-ready versions of most conference papers, since wrong font mappings lead to printing problems. While it is unfortunately not possible to solve the restriction to latin characters without modifying GNU R itself, the embedding of fonts can be solved easily: the plot PDF file simply has to be processed by GHOSTSCRIPT using *pdfwrite* as output device. The resulting new PDF file will get all required fonts embedded. Furthermore, the resulting file will also be compressed.

7. DEBUGGING

A non-trivial source code almost certainly contains bugs. For debugging of the RSPSIM model's implementation, we have made intensive use of VALGRIND [28,32]. In short, VALGRIND is a x86/x86_64 binary code interpreter that actually executes a program and keeps track of all memory accesses. Especially, it does not only remember which bit belongs to an allocated chunk of memory but also remembers which bit is still uninitialized.

That is, VALGRIND does not only detect accesses to invalid or already deallocated memory blocks but also warns when an uninitialized bit is used e.g. in a conditional branch. This category of errors is otherwise extremely difficult to discover: for example, a 16-bit variable is uninitialized, i.e. it contains a random value. While the probability is 65,535:1 that it contains a non-zero value, it may just contain 0 in an inappropriate moment and lead to a severe and almost untraceable (since difficult to reproduce) malfunction of the program. Furthermore, since VALGRIND keeps track of all memory allocations, it can also easily detect memory leaks. All errors found by VALGRIND are printed with function call stack trace as well as the corresponding source code file names and line numbers. So, it gets very easy to locate problems in the implementation of the model.

8. CONCLUSIONS

The goal of this paper has been an introduction to our Open Source tool-chain for the parametrization, distributed run execution, results post-processing and debugging of simulation models based on OMNeT++. Although initially being motivated by our RSerPool simulation model RSPSIM, it has been designed with model-independence in mind and is therefore applicable to arbitrary models.

RSerPool is a light-weight framework for server pool management, load balancing and failover handling. Since we have also developed the Open Source RSerPool prototype implementation RSPLIB as part of our research on RSerPool, our run distribution approach simply utilizes the infrastructure which is already provided by RSerPool. That is, it gets quite easy to efficiently parallelize simulation runs – RSerPool handles the pool maintenance, load balancing, session management and failover procedure. The tasks of parametrizing simulation runs as well as post-processing and

[5]A detailed parameter description can be found in [3].

Listing 3 An Example Plot Definition

```
simulationDirectory <- "wp1-hom-puToPERatioI"
...

# ====== Templates ================================================================
rspsim5PlotVariables <- list(
   # ------- System Utilization Template ------------------------------------------
   list("controller.SystemAverageUtilization",
        "Average Utilization [%]",
        "100.0 * data1$controller.SystemUtilization",
        "blue4",
        list("controller-SystemUtilization")),
   ...
)
...

# ====== Plots ====================================================================
plotConfigurations <- list(
   # ------- System Utilization Plot ----------------------------------------------
   list(simulationDirectory, paste(sep="", simulationDirectory, "-Utilization.pdf"),
        "Provider's Perspective", NA, NA, list(1,0),
        "puToPERatio", "controller.SystemAverageUtilization",
        "calcAppPoolElementSelectionPolicy", "jsToSC", ""),

   # ------- Handling Speed Plot --------------------------------------------------
   list(simulationDirectory, paste(sep="", simulationDirectory, "-HandlingSpeed.pdf"),
        "User's Perspective", NA, NA, list(0,1),
        "puToPERatio", "controller.SystemAverageHandlingSpeed",
        "calcAppPoolElementSelectionPolicy", "jsToSC", "")
)
```

plotting the results are handled by scripts based on GNU R. Finally, VALGRIND is utilized for debugging.

Currently, we are evaluating application checkpointing approaches for the suspension and resumption of long-time runs. Furthermore, we are also testing a XEN-based virtualization approach for security.

9. REFERENCES

[1] CONRAD, P., JUNGMAIER, A., ROSS, C., SIM, W.-C., AND TÜXEN, M. Reliable IP Telephony Applications with SIP using RSerPool. In *Proceedings of the State Coverage Initiatives, Mobile/Wireless Computing and Communication Systems II* (Orlando, Florida/U.S.A., July 2002), vol. X. ISBN 980-07-8150-1.

[2] DREIBHOLZ, T. Thomas Dreibholz's RSerPool Page, 2006.

[3] DREIBHOLZ, T. *Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture*. PhD thesis, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, Mar. 2007.

[4] DREIBHOLZ, T. Applicability of Reliable Server Pooling for Real-Time Distributed Computing. Internet-Draft Version 04, IETF, Individual Submission, Jan. 2008. draft-dreibholz-rserpool-applic-distcomp-04.txt, work in progress.

[5] DREIBHOLZ, T., COENE, L., AND CONRAD, P. Reliable Server Pooling Applicability for IP Flow Information Exchange. Internet-Draft Version 05, IETF, Individual Submission, Jan. 2008. draft-coene-rserpool-applic-ipfix-05.txt, work in progress.

[6] DREIBHOLZ, T., JUNGMAIER, A., AND TÜXEN, M. A new Scheme for IP-based Internet Mobility. In *Proceedings of the 28th IEEE Local Computer Networks Conference (LCN)* (Königswinter/Germany, Nov. 2003), pp. 99–108. ISBN 0-7695-2037-5.

[7] DREIBHOLZ, T., AND PULINTHANATH, J. Applicability of Reliable Server Pooling for SCTP-Based Endpoint Mobility. Internet-Draft Version 03, IETF, Individual Submission, Jan. 2008. draft-dreibholz-rserpool-applic-mobility-03.txt, work in progress.

[8] DREIBHOLZ, T., AND RATHGEB, E. P. Implementing the Reliable Server Pooling Framework. In *Proceedings of the 8th IEEE International Conference on Telecommunications (ConTEL)* (Zagreb/Croatia, June 2005), vol. 1, pp. 21–28. ISBN 953-184-081-4.

[9] DREIBHOLZ, T., AND RATHGEB, E. P. On the Performance of Reliable Server Pooling Systems. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN) 30th Anniversary* (Sydney/Australia, Nov. 2005), pp. 200–208. ISBN 0-7695-2421-4.

[10] DREIBHOLZ, T., AND RATHGEB, E. P. The Performance of Reliable Server Pooling Systems in Different Server Capacity Scenarios. In *Proceedings of the IEEE TENCON '05* (Melbourne/Australia, Nov. 2005). ISBN 0-7803-9312-0.

[11] DREIBHOLZ, T., AND RATHGEB, E. P. An Evalulation of the Pool Maintenance Overhead in Reliable Server Pooling Systems. In *Proceedings of the IEEE International Conference on Future Generation Communication and Networking (FGCN)* (Jeju

Island/South Korea, Dec. 2007), vol. 1, pp. 136–143. ISBN 0-7695-3048-6.

[12] Dreibholz, T., and Rathgeb, E. P. On Improving the Performance of Reliable Server Pooling Systems for Distance-Sensitive Distributed Applications. In *Proceedings of the 15. ITG/GI Fachtagung Kommunikation in Verteilten Systemen (KiVS)* (Bern/Switzerland, Feb. 2007), pp. 39–50. ISBN 978-3-540-69962-0.

[13] Dreibholz, T., and Rathgeb, E. P. Reliable Server Pooling – A Novel IETF Architecture for Availability-Sensitive Services. In *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)* (Sainte Luce/Martinique, Feb. 2008), pp. 150–156. ISBN 978-0-7695-3087-1.

[14] Dreibholz, T., Zhou, X., and Rathgeb, E. P. A Performance Evaluation of RSerPool Server Selection Policies in Varying Heterogeneous Capacity Scenarios. In *Proceedings of the 33rd IEEE EuroMirco Conference on Software Engineering and Advanced Applications* (Lübeck/Germany, Aug. 2007), pp. 157–164. ISBN 0-7695-2977-1.

[15] Eastlake, D., and Jones, P. US Secure Hash Algorithm 1 (SHA1). Informational RFC 3174, IETF, Sept. 2001.

[16] Eaton, J. Octave Home Page, 2003.

[17] Free Software Foundation. GNU Make, 2003.

[18] ITU-T. Introduction to CCITT Signalling System No. 7. Tech. Rep. Recommendation Q.700, International Telecommunication Union, Mar. 1993.

[19] Jungmaier, A., Rathgeb, E. P., and Tüxen, M. On the Use of SCTP in Failover-Scenarios. In *Proceedings of the State Coverage Initiatives, Mobile/Wireless Computing and Communication Systems II* (Orlando, Florida/U.S.A., July 2002), vol. X. ISBN 980-07-8150-1.

[20] Lei, P., Ong, L., Tüxen, M., and Dreibholz, T. An Overview of Reliable Server Pooling Protocols. Internet-Draft Version 05, IETF, RSerPool Working Group, Jan. 2008. draft-ietf-rserpool-overview-05.txt, work in progress.

[21] NS-2. The Network Simulator NS-2, 2003.

[22] OPnet Technologies. OPnet Modeler, 2003.

[23] Plank, J. S., Beck, M., Kingsley, G., and Li, K. Libckpt: Transparent Checkpointing under Unix. In *Proceedings of the USENIX Winter 1995 Technical Conference* (New Orleans, Louisiana/U.S.A., Jan. 1995), pp. 213–224.

[24] Pulinthanath, J. Zuverlässige Übertragung von IPFIX-Nachrichten mit der RSerPool-Architektur. Master's thesis, Universität Duisburg-Essen, Institut für Experimentelle Mathematik, Nov. 2007.

[25] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna/Austria, 2005. ISBN 3-900051-07-0.

[26] Rüngeler, I., Tüxen, M., and Rathgeb, E. P. Integration of SCTP in the OMNeT++ Simulation Environment. In *Proceedings of the 1st OMNeT++ Workshop* (Marseille/France, Mar. 2008). ISBN 978-963-9799-20-2.

[27] Seward, J. *bzip2 - A program and library for data compression*. Snowbird, Utah/U.S.A., Feb. 2005.

[28] Seward, J., and Nethercote, N. Using Valgrind to detect undefined value errors with bit-precision. In *Proceedings of the USENIX'05 Annual Technical Conference* (Anaheim, California/U.S.A., Apr. 2005), pp. 17–30.

[29] Stewart, R., Xie, Q., Stillman, M., and Tüxen, M. Aggregate Server Access Protcol (ASAP). Internet-Draft Version 18, IETF, RSerPool Working Group, Nov. 2007. draft-ietf-rserpool-asap-18.txt, work in progress.

[30] Tüxen, M., and Dreibholz, T. Reliable Server Pooling Policies. Internet-Draft Version 07, IETF, RSerPool Working Group, Nov. 2007. draft-ietf-rserpool-policies-07.txt, work in progress.

[31] Uyar, Ü., Zheng, J., Fecko, M. A., Samtani, S., and Conrad, P. Evaluation of Architectures for Reliable Server Pooling in Wired and Wireless Environments. *IEEE JSAC Special Issue on Recent Advances in Service Overlay Networks 22*, 1 (2004), 164–175.

[32] Valgrind Developers. Valgrind Home, 2005.

[33] Varga, A. *OMNeT++ Discrete Event Simulation System User Manual - Version 3.2*. Technical University of Budapest/Hungary, Mar. 2005.

[34] Williams, T., and Kelley, C. GNU Plot Homepage, 2003.

[35] Xie, Q., Stewart, R., Stillman, M., Tüxen, M., and Silverton, A. Endpoint Handlespace Redundancy Protocol (ENRP). Internet-Draft Version 18, IETF, RSerPool Working Group, Nov. 2007. draft-ietf-rserpool-enrp-18.txt, work in progress.

[36] Zhou, X., Dreibholz, T., and Rathgeb, E. P. A New Approach of Performance Improvement for Server Selection in Reliable Server Pooling Systems. In *Proceedings of the 15th IEEE International Conference on Advanced Computing and Communication (ADCOM)* (Guwahati/India, Dec. 2007), pp. 117–121. ISBN 0-7695-3059-1.

[37] Zhou, X., Dreibholz, T., and Rathgeb, E. P. Evaluation of a Simple Load Balancing Improvement for Reliable Server Pooling with Heterogeneous Server Pools. In *Proceedings of the IEEE International Conference on Future Generation Communication and Networking (FGCN)* (Jeju Island/South Korea, Dec. 2007), vol. 1, pp. 173–180. ISBN 0-7695-3048-6.

[38] Zhou, X., Dreibholz, T., and Rathgeb, E. P. Improving the Load Balancing Performance of Reliable Server Pooling in Heterogeneous Capacity Environments. In *Proceedings of the 3rd Asian Internet Engineering Conference (AINTEC)* (Phuket/Thailand, Nov. 2007), vol. 4866 of *Lecture Notes in Computer Science*, Springer, pp. 125–140. ISBN 978-3-540-76808-1.

[39] Zhou, X., Dreibholz, T., and Rathgeb, E. P. A New Server Selection Strategy for Reliable Server Pooling in Widely Distributed Environments. In *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)* (Sainte Luce/Martinique, Feb. 2008), pp. 171–177. ISBN 978-0-7695-3087-1.