Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 291 (2013) 3–14

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

SFERA: A Simulation Framework for the Performance Evaluation of Restart Algorithms in Service-Oriented Systems

Alexandra Danilkina[1]

Institute for Computer Science
Freie Universität Berlin
Berlin, Germany

and

Philipp Reinecke[2]

HP Labs
Bristol, UK

and

Katinka Wolter[3]

School of Computing Science
Newcastle University, UK

Abstract

In service-oriented systems, fault detection and localisation are not straightforward, and client-side fault-tolerance techniques are required to reduce the impact of faults on the quality of service experienced by the user. Restart is a well-known client-side technique for improving performance and service availability. With restart, tasks whose completion-time exceeds a timeout are re-issued by the client, with the goal of obtaining a shorter completion-time on the next attempt. Evaluation of restart should be performed by a combination of analysis, simulation, and measurement. In this paper we present the SFERA framework for simulation of restart in complex SOA systems. We illustrate SFERA features with an evaluation of the optimal restart timeout in a complex SOA system. We simulate a SOA system using different scenarios and model component response-times by phase-type distributions fitted to measurements from a SOA testbed. We observe and compare completion times for different scenarios.

Keywords: Fault-tolerance Mechanisms, Application-Level Restart, Service-Oriented Systems, Simulation Framework

[1] alexandra.danilkina@fu-berlin.de

[2] philipp.reinecke@hp.com

[3] katinka.wolter@ncl.ac.uk

doi:10.1016/j.entcs.2012.11.014

1 Introduction

Due to the growing importance of systems with Service-Oriented Architecture (SOA), the dependability of such systems becomes increasingly important. In complex systems like these detection and localisation of faults are not straightforward, and fault-tolerance mechanisms must be employed at different layers of the system. The restart method, where the client repeats service requests if no reply to the service request is received within a predefined time interval, is a simple and practical method for improving the service-availability and performance experienced by the client [18].

The impact of restart strategies can be evaluated analytically, in simulations, and in testbeds. In complex systems, simulation is often the preferred approach, as analytical methods usually cannot be applied, and experimentation is too expensive. In this paper, we describe our SFERA framework for simulation-based investigation of restart algorithms and their impact on request completion times in service-oriented systems. SFERA, the *Simulation Framework for performance Evaluation of Restart Algorithms*, allows simple and efficient modelling and simulation of restart in SOA systems. Using the framework, the effect of timeout strategies on the timing behaviour of the system, as seen by the client, can be evaluated for a large variety of timeout strategies, task semantics, workflows, and processing policies.

This paper is structured as follows: We first provide a brief introduction to the restart method on the application level. We then dicuss aspects of the evaluation of restart strategies in service-oriented systems. Sections 4 and 5 introduce the SFERA framework. Application of the framework is illustrated in section 6. Section 7 concludes the paper.

2 Application-Level Restart

With the restart method, clients repeat requests, in the hope of receiving a faster response from the system on the next trial. The time until the client restarts a request is called the restart timeout. The timeout interval determines the effect of restart: With a short timeout, system load increases, leading to longer completion times. In contrast, a very large value for the timeout restarts the request rarely, if at all and is equivalent to the completion time without restart. Restart is thus a timeout computation problem, where the optimal timeout has to be determined by a restart algorithm.

Two classes of restart algorithms can be distinguished: Adaptive algorithms collect completion times of tasks and compute the timeout with respect to these observations, whereas the timeout calculation by the non-adaptive algorithms is independent of the past (cf. [13]). The Fixed-Intervals algorithm, where requests are restarted after a constant amount of time, is an example for a non-adaptive algorithms. The QEST [15] and Jacobson/Karn (JK) [5][8] algorithms are adaptive. Detailed discussions of the algorithms in the context of restart are available in

e.g. [13][18].

Client requests may be abortable or non-abortable. In systems with abortable tasks previous tasks can be aborted before a new task is issued, while with non-abortable tasks the new request does not replace the old one. Client requests can also be classified by interarrival characteristic: Synchronous requests can be sent to the system only after the previous task is completed. Asynchronous requests can be simultaneously processed by the SOA.

These request characteristics may be combined, e.g. requests can be abortable and asynchronous, or abortable and synchronous. The different combinations have different implications for the possibility of overload and for the applicability of restart. In particular, synchronous, non-abortable requests are of no interest in the context of restart, since the restarted task is processed by the system only after the completion of the previous one. In this situation, the waiting time cannot be reduced through restart.

3 Evaluation of Restart in Service-Oriented Systems

Our research is concerned with the evaluation of dependability phenomena of distributed systems. Different performance and reliability measurements can be evaluated on different levels of abstraction. We observe systems from the client's point of view. Typically the client does not know the implementation and architectural details of requested services. Clients are only able to collect the information about timing behaviour of their requests, such as start times and completion times, and can apply restart using different algorithms. Clients do not communicate with each other.

We consider clients requesting a service, which in turn may be composed from several independent services. Each service is therefore assumed to be able to represent both a service and a client requesting other services. Individual services of a composed service may be called in various ways; the exact specification of the order of requests is referred to as the workflow.

Restart addresses faults that manifest in the completion-time behaviour of tasks. In a service-oriented system, many common faults such as system failures, network faults, or system overload result in higher completion times, including an infinite completion time in the case of a failure. In evaluating restart in SOA systems we therefore focus on completion times. Restart algorithms may be evaluated at various levels of detail, including measurements in test-beds, simulation, and analytical approaches.

In the measurement-based approach, one first implements a testbed comprising clients, servers, and restart algorithms, and then measures completion times with and without restart. Tools for automatic testbed generation such as PUPPET [2] and GENESIS [7] can be employed in setting up a testbed. With this approach, realistic system completion times and timeouts can be collected. The experiments are limited by available hardware and by long runtime.

The analytical approach describes and evaluates the system with and without

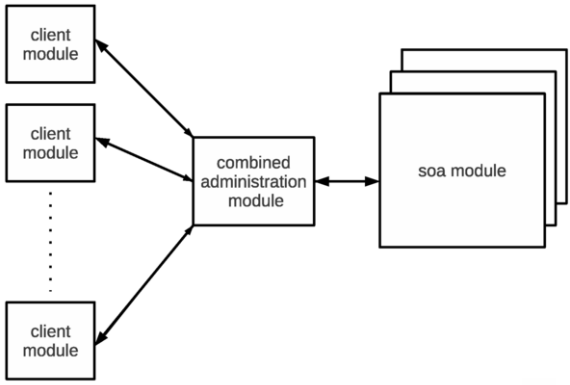

Fig. 1. Simulation framework architecture.

restart using stochastic models (see e.g. [16][18]). This approach gives general results for the timing behaviour. The disadvantage of this method is the difficulty in formalising realistic models for system behaviour and complexity of the resulting models.

A framework for simulation models of service-oriented architectures was presented in [1]. The framework combines an extended version of process chains to describe the components of SOA and quantitative specifications at higher levels. The aim of the framework is to be able to simulate SOA system architecture and the network connecting independent services in detail. The framework in [1] focusses on precise modelling of systems implementing low-level details. While restart may be evaluated using this framework, scalability to large numbers of clients is limited by the highly-detailed approach.

In contrast to the approach in [1], we employ a high-level approach, where we model the impact of faults and low-level network and system details by service-time distributions. In order to evaluate restart mechanisms, we develop a simulation framework for collecting completion times for client requests. The framework supports large numbers of clients requesting a service in various ways. Requests can be abortable and non-abortable, as well as synchronous and asynchronous. We implement three restart algorithms and enable the user to build and evaluate models of systems with different workflows.

4 SFERA Design

The SFERA simulation framework provides a method to study completion times of a service-oriented system under various restart policies, assuming that a model for the timing behaviour of the services without restart is available. SFERA uses an abstract modelling and simulation approach, in order to provide flexible and efficient evaluation. The architecture of the framework is shown in Figure 1. The framework consists of three modules: *Client Modules* simulate users in the system. Clients

generate requests, apply restart, and collect completion times. *SOA Modules* model services of the SOA system. SOA Modules receive requests from clients and send responses back. Service on the requests is simulated by appropriate delays according to a service-time distribution. Typically, the service-time distribution in a SOA module models the behaviour of the system for one client. As additional parallel clients cause growing system load, a load model for the impact of multiple clients is also part of the SOA Module. Additionally, SOA Modules can model various service policies as well as phenomena such as aging and breakdown, and mechanisms such as rejuvenation (cf. [18] for an in-depth discussion of aging phenomena and rejuvenation mechanisms). The *Combined Administration* module manages requests of parallel clients by forwarding requests and responses between the clients and the SOA module. This module is used not only to help structure the simulation code but also implements workflows.

The framework allows to configure a system with various scenarios. A scenario is defined by the client type, the SOA module type and their combination to form the entire system. The SOA module type is determined by the service-time distribution and by the service policy. The client type is defined by the restart algorithm and the arrival process requests. The framework allows to combine all modules when configuring the system. Both clients and SOA modules can be set up in parallel.

Note that the focus of SFERA is on evaluating the timing behaviour of a SOA system with different numbers of clients, as observed by the user. We therefore model request/response exchanges by simple messages, rather than by simulating network connections in detail. While it may be argued that detailed simulation of network connections could improve accuracy of the results, it would also increase simulation complexity and simulation run-times. In order to obtain simple and efficient simulations, we capture all low-level behaviour in stochastic fault-models (cf. [14]). The SFERA architecture therefore does not map an original system in detail to the simulation, but its abstraction level is adequate for the performance evaluation of restart mechanisms.

5 Implementation details of SFERA

In this section we discuss several implementation details of SFERA, which may be of interest for the user of the framework.

SFERA is implemented using the OMNeT++ discrete-event simulation framework [17], and OMNet++'s NED language is used to set up simulation models. As described in Section 4, SFERA provides three different types of modules for the composition, each of which could provide several different methods. For instance, a Client Module may support synchronous, abortable requests and use the Fixed Intervals restart algorithm, or a SOA Module may exhibit both aging and rejuvenation. We summarise the characteristics currently supported in Table 1.

Table 1
SFERA Modules Summary.

Module Name	Characteristics
Client Module	– arrival process: synchronous/asynchronous requests – restart strategies: FI, QEST, JK – request class: abortable, non-abortable – deadline
Administration Module	workflows: parallel, in sequence, mixed
SOA Module	– aging: explicit, implicit – rejuvenation: different policies – service process: FIFO queue, synchronous – number of servers: infinite, single – service policy: slots-model, processor sharing – service distribution: standard statistical, phasetype

5.1 Client Module

We implemented two types of client modules, which differ in the interarrival characteristics of the request. The first type is called sequential source and sends synchronous requests to the system. The second implements asynchronous requests and we call this source parallel. By default, both services use a Poisson interarrival process as the interarrival process of requests. The sequential source waits for a response before sending the next request to the server. Jobs arriving while the previous job is in processing are stored in a local queue in the client. In contrast, the parallel source sends new requests directly to the server. Consequently, with a sequential source there will never be two jobs from the same client in the system, while with a parallel source several jobs may be processed in parallel.

The client modules that are currently available implement the three restart mechanims discussed in Section 2; additionally, a 'no restart' policy is supported. Restart strategies are implemented in a modular way, using the oracle approach described in [12].

Sources collect completion times either as raw data files or as statistics using the objects provided by OMNeT++. The data is written to one output file per client, which enables later investigation of the data using external scripts. For performance reasons, SFERA does not utilise the built-in data storage and analysis facilities of OMNeT++.

5.2 SOA Module

In the implementation, SOA Modules are referred to as work servers. A work server models a SOA service as a black box that receives requests and sends responses, modelling processing by delays between the request and the response. The work server makes the simulation framework reusable and flexible, because it can simulate any system once the service-time distribution is known. Work servers can obtain their service times from one of the statistical distributions provided by OMNeT++ or draw them from a phase-type distribution using the Libphprng library [11]. More general stochastic processes could be included by using the module described in [9].

We implemented three types of work servers, referred to as the simple, advanced, and processor sharing work server.

The simple work server does not model load on the system and simply draws service-times from the specified distribution. The advanced work server has a model for load generated by multiple simultaneous requests. We use a simple 'slot' model for capturing the impact of load: Each client is assigned a slot in the server, and the available service capacity is shared between clients based on the number of used slots and a constant factor. E.g., if one job is in service and a new job arrives, the service time for the new job is multiplied by the factor 2, since only one half of the service capacity is available for the new job. Note that this model captures the effect of sharing system resources only very roughly, since dynamics are not reflected at all. However, the model proves to be very efficient in simulation, as compared to processor sharing (cf. [6] e.g.). Finally, we implement the processor sharing policy. The service module monitors the number of jobs in service and reschedules completion events accordingly upon changes.

5.3 Administration Module

The Administration module forwards requests and responses between clients and servers, and manages the execution of workflows. Workflows describe the sequence of work servers that should process the client request in order to build the final answer. We distinguish between the parallel, sequential, and mixed workflows. In the parallel workflow the administration module sends the request simultaneously to all of the work servers. In the sequential workflow the request is sent successively to the work servers. The mixed workflow allows to combine the first two types.

6 A Case Study Using SFERA

In this section we present a case study where we use SFERA in the evaluation of restart algorithms in two scenarios. The first scenario consists of parallel clients requesting one service, while in the second scenario a single client requests a complex service, which is implemented as a workflow. Our goal is to compare the impact of different restart algorithms on the timing behaviour of the system and to provide an example of using SFERA for practical issues. For the evaluation of the algorithms completion times need to be collected and the statistical properties of the data sets

must be analysed and compared. The case study is structured as follows: First we discuss the setup of the simulations. We then present the parameters used in the evaluation, followed by a discussion of the results that can be obtained.

6.1 Simulation Setup

In the first simulation we want to model a scenario where a large number of clients access the same service in parallel. We assume that client requests belong to one of two classes, which differ in their service demand. This scenario can be set up in SFERA using several client modules, one work server, and an administration module to forward requests and responses between the clients and the work server. The second simulation uses a single client requesting a complex service, which is implemented as mixed workflow. This experiment can be set up using SFERA's client module, the administration module implementing the workflow and three work servers. In order to obtain a realistic simulation, we need an appropriate model for the service times in work servers. While one may use any of the theoretical distributions provided by OMNeT++, the following experimental approach yields more realistic service times: We first set up a SOA testbed with and without fault injection. We then measured completion times with one client and no restart. The distribution of completion times was used as service-time distributions in our evaluation.

The testbed was composed of four web services with real functionality providing a combined service. Different services accomplish diverse tasks and occupy different resources. Individual services were deployed separately on dedicated machines, connected by a wired network. Two experiments were conducted - with and without injected packet-loss between the client and the testbed service. Because completion times are measured in a realistic system, they build a solid basis for the evaluation of system times.

Packet loss is a common fault in the transport layer of network-based systems. Packet loss can be caused by overload or unstable connections [19][14]. In one of the experiments we inject packet loss using the *NetemCG* Linux kernel module [3][10].

We measured in both experiments in each case 5000 completion-time samples. Completion times were then approximated with acyclic phase-type distributions using the PhFit phase-type fitting tool [4]. The output of PhFit is a distribution which is an accurate model of the distribution of the collected completion times. This model was used in our evaluation as models for the service-times in work servers.

6.2 Scenario Definition and SFERA Settings

We use in both simulations the parallel client of SFERA, which generates asynchronous, abortable requests. We chose exponential interarrival times with a mean of 4 seconds. The number of parallel clients in the first simulation was set to 1, 2, 20, 50, and 100. We use the 'slots' service policy described in Section 5 to model the simultaneous service of parallel requests. The experiment using a mixed workflow

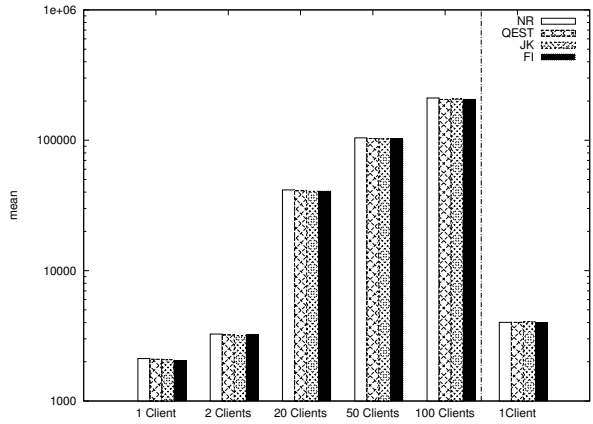

Fig. 2. Means of completion times for scenarios with parallel clients on the left and with a complex service on the right (logarithmic scale; NR – No Restart, FI – Fixed Intervals, JK – Jacobson/Karn).

simulates one parallel client. The workflow is combined from three work servers. First, two servers using service times without faults are called in parallel. Once the administrative module receives answers from both parallel work servers, the next work server is called. This work server represents a faulty service and uses a model of service times from a testbed with packet loss. We ran simulations using the Fixed Intervals, Jacobson Karn, and QEST restart algorithms. Appropriate parameters for the restart algorithms were estimated based on the completion times from the testbed: The timeout for the Fixed Intervals algorithm was set around the mean of the observed completion times. The parameters for the histogram for QEST were adjusted based on empirical quantiles and variance. The Jacobson/Karn algorithm was set to its default values [5][8].

6.3 Results

The next step of the experiment is to evaluate statistical properties of the experiments with restart for both experiments. First we analyse the empirical mean of completion times through all scenarios. We see the mean as an indicator for the impact of restart algorithms on the other statistical measures.

The left side of the Figure 2 shows that the mean can be slightly reduced by each of the analysed restart methods for the scenario with parallel clients. But the impact on the mean by restart is greater in scenarios with a large number of parallel clients. The right side of Figure 2 shows the scenario with a complex service and one client: Only the FI-Algorithm can reduce the mean.

The evaluation of the 95%-quantile is presented in Figure 3. As we can see, not every analysed restart algorithm can reduce the quantiles: the JK and QEST algorithms increase the 95%-quantile for scenarios with 1, 2, 20 and 50 clients. In these scenarios, only the FI algorithm could reduce this metric. The same tendency one can observe for the quantiles of completion times of the complex service. It should be noted that both mean and the 95%-quantile can be reduced only marginally in all scenarios.

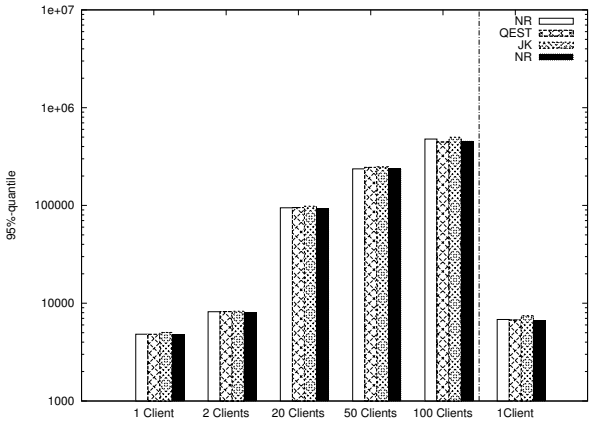

Fig. 3. 95%-quantiles for scenarios with parallel clients on the left and a complex service on the right (logarithmic scale; NR – No Restart, FI – Fixed Intervals, JK – Jacobson/Karn).

To summarise the evaluation, none of three restart algorithms can significantly reduce statistical properties of services times. This can be explained by the small *scv* (squared coefficient of variation) of the testbed data sets. The data set without faults has $scv = 1.195017$, the service times from the experiment with packet loss have $scv = 1.79$. Nevertheless, these experiments show the practical side of using SFERA, as complex simulations with parallel clients in first configuration and the combed workflow in the second could be evaluated. Obtaining a similar result for the scenarios from a testbed setup would incur substantial hardware and software costs. Note that various runs were necessary in order to provide an appropriate optimal timeout for the FI restart algorithm. The figures above only show results for the optimal timeout.

7 Conclusion

The SFERA framework provides a strong background for the evluation of timing behaviour of distributed systems where services are requested by clients applying restart. While our focus in this paper was on service-oriented systems, SFERA is not limited to such systems. The framework can be used to evaluate restart in other scenarios where similar technologies are employed and timing behaviour depends on the correct operation of complex layers of systems. For instance, one future application area is in Cloud Computing, where services are requested from highly complex servers whose completion-times depend not only on the network or the load, but also on the virtualisation layer.

SFERA is a versatile tool for simulation of both general and complex configurations of services and various scenarios. The modular design of the framework allows experienced user to design comprehensive experiments using e.g. PH-distributed service times; less experienced users can simply compose a service oriented architecture with client-side restart.

We are currently preparing SFERA for distribution. Furthermore, we aim to extend the functionality of SFERA modules and provide new modules. Also the

existing components will be optimised for faster simulation.

Acknowledgement

This work was supported by DFG grants Wo 898/2-1 and Wo 898/3-1. The work presented here was performed while Philipp Reinecke was at Freie Universität Berlin.

References

[1] F. Bause, P. Buchholz, J. Kriege, and S. Vastag. A Framework for Simulation Models of Service-Oriented Architectures. In S. Kounev, I. Gorton, and K. Sachs, editors, *Performance Evaluation: Metrics, Models and Benchmarks*, volume 5119 of *Lecture Notes in Computer Science*, pages 208–227. Springer Berlin / Heidelberg, 2008.

[2] A. Bertolino, G. D. Angelis, and A. Polini. A QoS Test-Bed Generator for Web Services. In L. Baresi, P. Fraternali, and G.-J. Houben, editors, *ICWE*, volume 4607 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2007.

[3] M. Dräger. Entwurf und Implementierung eines Moduls zur stochastischen Fehlerinjektion für IP-Netzwerke gemäß eines erweiterten Gilbert-Modells. Bachelor thesis, Freie Universität Berlin, 2011. (in German).

[4] A. Horváth and M. Telek. PhFit: A General Phase-Type Fitting Tool. *Field, T.,Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS*, 2324:82–91, 2002.

[5] V. Jacobson. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[6] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley, 1991.

[7] L. Juszczyk, H.-L. Truong, and S. Dustdar. GENESIS - A Framework for Automatic Generation and Steering of Testbeds of Complex Web Servicesnet. In *Proc. 13th IEEE International Conference on Engineering of Complex Computer Systems ICECCS 2008*, pages 131–140, March 31 2008–April 3 2008.

[8] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. *ACM Trans. Comput. Syst.*, 9:364–373, November 1991.

[9] J. Kriege and P. Buchholz. Simulating Stochastic Processes with OMNeT++. In *Proceedings of the 4th International OMNeT++ Workshop (OMNeT++ 2011)*. ICST, 2011.

[10] P. Reinecke, M. Dräger, and K. Wolter. NetemCG IP Packet-Loss Injection using a Continuous-Time Gilbert Model. Technical Report TR-B-11-0, Freie Universitt Berlin, September 2011.

[11] P. Reinecke and G. Horváth. Phase-type Distributions for Realistic Modelling in Discrete-Event Simulation. In *OMNeT++ Workshop 2012*. ACM, 2012. To appear.

[12] P. Reinecke, A. P. A. Van Moorsel, and K. Wolter. A Measurement Study of the Interplay Between Application Level Restart and Transport Protocol. In M. Malek, M. Reitenspieß, and J. Kaiser, editors, *Service Availability. Proceedings of the First International Service Availability Symposium*, volume 3335 of *LNCS*, pages 86–100. Springer, May 2004.

[13] P. Reinecke, A. P. A. Van Moorsel, and K. Wolter. The fast and the fair: A fault-injection-driven comparison of restart oracles for reliable web services. In *3rd International Conference on the Quantitative Evaluation of SysTems (QEST) , pp. 375-384, Riverside, CA, USA, IEEE-CS Press, Sept. 2006.*, pages 375–384. IEEE-CS Press, 2006.

[14] P. Reinecke, K. Wolter, and M. Malek. A Survey on Fault-Models for QoS Studies of Service-Oriented Systems. Technical Report B-2010-02, Freie Universität Berlin, February 2010.

[15] A. P. A. Van Moorsel and K. Wolter. Analysis and algorithms for restart. In *In Proc. 1st International Conference on the Quantitative Evaluation of Systems (QEST*, pages 195–204. IEEE Computer Society, 2004.

[16] A. P. A. Van Moorsel and K. Wolter. Analysis of Restart Mechanisms in Software Systems. *IEEE Trans. on Software Eng*, 32:547–558, 2006.

[17] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, June 2001.

[18] K. Wolter. *Stochastic Models for Fault Tolerance - Restart, Rejuvenation and Checkpointing.* Springer Verlag, 2010.

[19] Y. Zhang, V. Paxson, and S. Shenker. The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. Technical report, ACIRI, 2000.