
# DEPEND 2012

The Fifth International Conference on Dependability

ISBN: 978-1-61208-212-7

August 19-24, 2012

Rome, Italy

**DEPEND 2012 Editors**

Syed Naqvi, CETIC, Belgium

Petre Dini, Concordia University, Canada / China Space Agency Center - Beijing, China

# DEPEND 2012

# Foreword

The Fifth International Conference on Dependability [DEPEND 2012], held between August 19-24, 2012 in Rome, Italy, provided a forum for detailed exchange of ideas, techniques, and experiences with the goal of understanding the academia and the industry trends related to the new challenges in dependability on critical and complex information systems.

Most of critical activities in the areas of communications (telephone, Internet), energy & fluids (electricity, gas, water), transportation (railways, airlines, road), life related (health, emergency response, and security), manufacturing (chips, computers, cars) or financial (credit cards, on-line transactions), or refinery& chemical systems rely on networked communication and information systems. Moreover, there are other dedicated systems for data mining, recommenders, sensing, conflict detection, intrusion detection, or maintenance that are complementary to and interact with the former ones.

With large scale and complex systems, their parts expose different static and dynamic features that interact with each others; some systems are more stable than others, some are more scalable, while others exhibit accurate feedback loops, or are more reliable or fault-tolerant.

Inter-system dependability and intra-system feature dependability require more attention from both theoretical and practical aspects, such as a more formal specification of operational and non-operational requirements, specification of synchronization mechanisms, or dependency exception handing. Considering system and feature dependability becomes crucial for data protection and recoverability when implementing mission critical applications and services.

Static and dynamic dependability, time-oriented, or timeless dependability, dependability perimeter, dependability models, stability and convergence on dependable features and systems, and dependability control and self-management are some of the key topics requiring special treatment. Platforms and tools supporting the dependability requirements are needed.

As a particular case, design, development, and validation of tools for incident detection and decision support became crucial for security and dependability in complex systems. It is challenging how these tools could span different time scales and provide solutions for survivability that range from immediate reaction to global and smooth reconfiguration through policy based management for an improved resilience. Enhancement of the self-healing properties of critical infrastructures by planning, designing and simulating of optimized architectures tested against several realistic scenarios is also aimed.

To deal with dependability, sound methodologies, platforms, and tools are needed to allow system adaptability. The balance dependability/adaptability may determine the life scale of a complex system and settle the right monitoring and control mechanisms. Particular challenging issues pertaining to context-aware, security, mobility, and ubiquity require appropriate mechanisms, methodologies, formalisms, platforms, and tools to support adaptability.

We take here the opportunity to warmly thank all the members of the DEPEND 2012 Technical Program Committee, as well as the numerous reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to DEPEND 2012. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the DEPEND 2012 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that DEPEND 2012 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the field of dependability.

We are convinced that the participants found the event useful and communications very open. We also hope the attendees enjoyed the historic charm Rome, Italy.

**DEPEND 2012 Chairs:**

Marcello Cinque, University of Naples Federico II, Italy
Petre Dini, Concordia University, Canada / China Space Agency Center - Beijing, China
Sergio Pozo Hidalgo, University of Seville, Spain
Syed Naqvi, CETIC, Belgium
Manuel Gil Perez, University of Murcia, Spain
Reijo Savola, VTT Technical Research Centre of Finland, Finland
Michiaki Tatsubori, IBM Research Tokyo, Japan
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany
Timothy Tsai, Hitachi Global Storage Technologies, USA
Szu-Chi Wang, National Ilan University, Taiwan
Piyi Yang, Wonders Information Co., Ltd., China

# DEPEND 2012

## Committee

**DEPEND Advisory Chairs**

Reijo Savola, VTT Technical Research Centre of Finland, Finland
Sergio Pozo Hidalgo, University of Seville, Spain
Manuel Gil Perez, University of Murcia, Spain
Petre Dini, Concordia University, Canada / China Space Agency Center - Beijing, China

**DEPEND 2012 Industry Liaison Chairs**

Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

**DEPEND 2012 Research/Industry Chair**

Michiaki Tatsubori, IBM Research Tokyo, Japan

**DEPEND 2012 Special Area Chairs**

**Cross-layers dependability**
Szu-Chi Wang, National Ilan University, Taiwan

**Hardware dependability**
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany

**Empirical assessments**
Marcello Cinque, University of Naples Federico II, Italy

**Security and Trust**
Syed Naqvi, CETIC, Belgium

**DEPEND 2012 Technical Program Committee**

Murali Annavaram, University of Southern California, USA
Afonso Araújo Neto, University of Coimbra, Portugal
José Enrique Armendáriz-Iñigo, Universidad Pública de Navarra, Spain
Steffen Bartsch, TZI - Universität Bremen, Germany
Jorge Bernal Bernabé, University of Murcia, Spain
Andrey Brito, Universidade Federal de Campina Grande, Brazil
Lasaro Camargos, Federal University of Uberlândia, Brazil
Juan Carlos Ruiz, Universidad Politécnica de Valencia, Spain
Antonio Casimiro Costa, University of Lisbon, Portugal

Simon Caton, Karlsruhe Institute of Technology (KIT), Germany
Zhe Chen, Nanjing University of Aeronautics and Astronautics, China
Marcello Cinque, University of Naples Federico II, Italy
Domenico Cotroneo, Università di Napoli Federico II, Italy
Rubén de Juan Marín, Universidad Politécnica de Valencia, Spain
Vincenzo De Florio, University of Antwerp, Belgium & IBBT, Belgium
Nicola Dragoni, Technical University of Denmark - Lyngby, Denmark
Diana El Rabih, Université Paris 12, France
Laila El Aimani, Technicolor, Security & Content Protection Labs., Germany
Alexander Felfernig, TU - Graz, Austria
Nuno Ferreira Neves, University of Lisbon, Portugal
Francesco Flammini, Ansaldo STS, Italy
Cristina Gacek, City University London, United Kingdom
Manuel Gil Perez, University of Murcia, Spain
Michael Grottke, University of Erlangen-Nuremberg, Germany
Nils Gruschka, NEC Laboratories Europe - Heidelberg, Germany
Bjarne E. Helvik, The Norwegian University of Science and Technology (NTNU) - Trondheim, Norway
Jiankun Hu, Australian Defence Force Academy - Canberra, Australia
Neminath Hubballi, Infosys Lab Bangalore, India
Arshad Jhumka, University of Warwick - Coventry, UK
Yoshiaki Kakuda, Hiroshima City University, Japan
Hui Kang, Stony Brook University, USA
Aleksandra Karimaa, Turku University/TUCS and Teleste Corporation, Finland
Dong-Seong Kim, University of Canterbury, New Zealand
Ezzat Kirmani, St. Cloud State University, USA
Seah Boon Keong, MIMOS Berhad, Malaysia
Abdelmajid Khelil, TU-Darmstadt, Germany
Kenji Kono, Keio University, Japan
Israel Koren, University of Massachusetts - Amherst, USA
Mani Krishna, University of Massachusetts - Amherst, USA
Patrick Lanigan, FeDex, USA
Mikel Larrea, University of the Basque Country - UPV/EHU, Spain
Inhwan Lee, Hanyang University - Seoul, Korea
Matthew Leeke, University of Warwick, UK
Paolo Lollini, University of Firenze, Italy
Miroslaw Malek, Humboldt-Universitaet zu Berlin, Germany
Rivalino Matias Jr., Federal University of Uberlandia, Brazil
Manuel Mazzara, Newcastle University, UK / UNU-IIST, Macau
Per Håkon Meland, SINTEF ICT, Norway
Francesc D. Muñoz-Escoí, Universitat Politècnica de València, Spain
Jogesh K. Muppala, The Hong Kong University of Science and Technology, Hong Kong
Jun Na, Northeastern University, China
Syed Naqvi, CETIC, Belgium
Sarmistha Neogy, Jadavpur University, India
Mats Neovius, Åbo Akademi University - Turku, Finland
Hong Ong, e-Manual System Sdn Bhd, Malaysia
Anne-Cécile Orgerie, ENS de Lyon, France / University of Melbourne, Australia
Aljosa Pasic, ATOS Origin, Spain

Wolfgang Pree, University of Salzburg, Austria
Felix Salfner, SAP Innovation Center - Potsdam, Germany
Reijo Savola, VTT Technical Research Centre of Finland, Finland
Dimitrios Serpanos, University of Patras & ISI, Greece
Navjot Singh, Avaya Labs Research, USA
Komminist Sisai, Fondazione Bruno Kessler, Italy
Kuo-Feng Ssu, National Cheng Kung University, Taiwan
Vladimir Stantchev, Berlin Institute of Technology, Germany
Neeraj Suri, TU-Darmstadt, Germany
Oliver Theel, University Oldenburg, Germany Sergio Pozo Hidalgo, University of Seville, Spain
Kishor Trivedi, Duke University - Durham, USA
Peter Tröger, Hasso Plattner Institute / University of Potsdam, Germany
Elena Troubitsyna, Aabo Akademi -Turku, Finland
Timothy Tsai, Hitachi Global Storage Technologies, USA
Marco Vallini, Politecnico di Torino, Italy
Ángel Jesús Varela Vaca, University of Sevilla, Spain
Bruno Vavala, Carnegie Mellon University, USA | University of Lisbon, Portugal
Hironori Washizaki, Waseda University, Japan
Hiroshi Yamada, Keio University, Japan
Piyi Yang, University of Shanghai for Science and Technology, China
Hee Yong Youn, Sungkyunkwan University, Korea

**Copyright Information**

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

**Table of Contents**


CGFAIL: A New Approach for Simulating Computer Systems Faults 1
*Martin Groessl*

Design of Dependable Systems: An Overview of Analysis and Verification Approaches 4
*Jose Ignacio Aizpurua Unanue and Enaut Muxika Olasagasti*

Dependable Design: Trade-Off Between the Homogeneity and Heterogeneity of Functions and Resources 13
*Jose Ignacio Aizpurua Unanue and Enaut Muxika Olasagasti*

Security Control Variations Between In-house and Cloud-based Virtualized Infrastructures 18
*Ramaswamy Chandramouli*

Towards a State driven Workload Generation Framework for Dependability Assessment 25
*Domenico Cotroneo, Francesco Fucci, and Roberto Natella*

Improving Consumer Satisfaction Through Building an Allocation Cloud 31
*Kuo Chen Wu, Hewijin Christine Jiau, and Kuo-Feng Ssu*

Design and Implementation of Cloud-based Application for Cloud Provider System with SSL Accelerator and Content Redirection 38
*Boon Keong Seah*

# CGFAIL: A New Approach for Simulating Computer Systems Faults

Martin Groessl
*Heidelberg Institute for*
*Theoretical Studies (HITS)*
*Heidelberg, GERMANY*
*Email: Martin.Groessl@h-its.org*

***Abstract*—An established method for emulation of faults in computer systems is fault injection. The application of such methods, typically, requires an extension of the operation system by special drivers. Here, a new approach for simulating a special kind of failure models, so called resource faults, is presented. The approach is currently directly supported from LINUX operation system and was tested with different distributions and architectures (X86/X64). The objective is to simulate fault models without affecting the normal operation of the computer system. Additionally, the method enables system developers to test their software under different resources conditions.**

***Keywords*-Software Dependability; Failure stimulation**

## I. INTRODUCTION

New computer systems composed of multiple processors, an amount of memory and shared resources build the backbone of modern information infrastructure. Testing of applications which run on such systems under realistic conditions is a quiet difficult job. Especially hardening, such software for fault tolerance [1], requires simulation of faults during testing. This is achieved by using fault injection techniques.

Depending on fault level different approaches have to be applied to stimulate the software under test. Some injection techniques modify the software under test (SuT) others necessitate an extension of the operation system by special customized drivers. A modification of the SuT leads to a deviating operation behavior, e.g., the timing behavior is changed. Especially by certified software which has to be tested under real conditions that should fulfill the operation specification it's impossible to use such approaches.

The goal in this paper is to present a technique for fault tolerance evaluation without modifying the SuT or the operating system. Here, standard drivers and libraries provided by the operating system are used to simulate faulty behavior.

The structure of this paper is as follows: Section 2 describes related research in the field of fault injection. Section 3 discusses the CGFAIL approach. An analysis of CGFAIL is presented in Section 4 which includes the supported fault classes and implementation details from a prototype. Finally, Section 7 concludes the paper.

## II. RELATED WORK

Fault generation is currently realized depending on the chosen fault model in simulation or fault injection in hardware / software. An overview of several software-based approaches is published in [2]. Fault injection on physical level which covers hardware faults with different constraints is presented in [3]. An approach for firmware level fault injection is pointed out in [4]. The point of action is based on a BIOS extension the Extensible Firmware Interface (EFI). Embedded in the EFI driver fault injection routines are located.

Software implemented fault injection (SWIFI) is an established method to emulate several hardware faults by programmatic changes in computer systems. A restriction for this technique is that only fault locations accessible by software are manipulatable. On the other hand SWIFI avoids permanent damage of hardware or usage of special stimulation hardware devices. A software-oriented fault injection framework which uses software traps to control the injection process is FERRARI [5]. Software traps are triggered by program counter when it points to the desired program locations or by a timer. When traps are triggered, the trap handling routines inject faults at the specific fault locations, like CPU, memory and bus.

FTAPE (Fault Tolerance And Performance Evaluator) [6] is a software tool that generates synthetic workloads that produce a high level of CPU, memory, and I/O activity and injects corresponding faults according to an injection strategy. Faults are injected based on this workload activity in order to ensure a high level of fault propagation. Xception [7] uses the advanced debugging and performance monitoring features existing in most of the modern processors to inject faults by software. Additionally the activation of the faults and their impact on the target system behavior is monitored. Faults injected by Xception can affect any process running on the target system (including the kernel), and it is possible to inject faults in applications for which the source code is not available.

Most SWIFT approaches require an extension of system software by special drivers. Alternatively, the application under test has to run in a special trace mode and depth

 ISBN: 978-1-61208-212-7 

knowledge of the applications structure is necessary.
The new concept supports simulation of resource faults based on standard OS drivers and libraries. Additionally, the behavior of the operation system and simultaneously running applications is not affected.

## III. Approach

A feature of modern LINUX operation systems is integrated support for resource management. An example for such a mechanism is CGOUPS (Control Groups) [8]. A direct supported by the kernel is provided see Figure 1. An advantage of CGROUPS is that support for resource limiting, prioritization, isolation, accounting control of resources is provided. Resource limitation on group level is motivated by not exceeding a predefined amount of provided resources. Such a restriction can be interpreted form a different point of view as a kind of sandboxing resources for applications. The isolation of CGROUPS provides a restrictive way to seal off provided resources for each individual group among each other and form global available. Additionally, the accounting which measure consume of resources from certain systems is a feature which offers an individual control of resources. An advantage of resource accounting compared to system measurement tools (top, htop) for the Linux kernel is the more precise resolution. In general, Linux kernel measurement tools update is one second.
CGROUPS was established in many LINUX distributions in 2007 and is usable without any kernel modifications. Neither the installation of special drivers is required. Some enhancements for additional hardware and resource support were done in 2010. Henceforward, this enables memory management and individual I/O device control without installing special drivers or modifying the kernel.

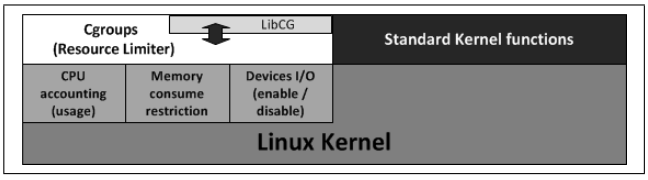


Figure 1. Overview CGROUPS in Linux kernel

A goal which is achievable by limitation of resources in combination with load generators is the emulation of real operational scenarios beyond that due to a dynamic reduction of provided resources a simulation of special fault models is feasible. A dynamic modification of CGROUP parameters during system runtime which is supported by an API-library (Libcg) enables a resizing of the sandbox. This leads to coverage of scenarios like a decreasing of parameter values beyond used amount and enables application developers to test their software under different environmental conditions including some borderline cases without modification of the operating system or the target application itself. The goal is

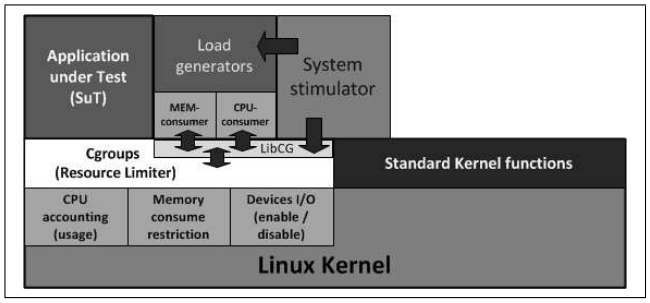


Figure 2. Sandbox regulator (CGROUPS) with load generator

reached by adjusting the amount of available resources as a resource sandboxing combined with load generation which run in the sandbox (see Figure 2). In such a way, simulations of borderline situation are practicable without stressing the computer system or even driving it into an abnormal state. A degrading of resources, like available memory, during runtime leads not only to realistic operational scenarios furthermore it's possible to drive an application to limiting cases. An example for such a behavior is a decreasing memory consume as brought about by memory leaks. In respect of mentioned failure scenarios the name CGFAIL was derived from CGROUPS. Typically abnormal memory consume drives the whole computer system into swap or even into crash. The application of resource sandboxes enables such simulation without affecting operation systems behavior. Borderline cases like Out-Of-Memory (OOM) are enforceable.

## IV. Analysis

In this section, an overview of up to date supported CGFAIL capabilities is presented. Depending on the controlled resources a derivation to identify several failure scenarios which are emulateable is done. Additionally the properties of SWIFI are set in relation with CGFAIL. CGROUPS support currently resource management for CPU, memory and devices. Thus only erroneous states based on this hardware parts are generateable. Driving these resources to limiting cases results in emulation of following presented fault classes:

- Low CPU resource availability
- Low memory availability
- Out of memory
- Unavailable devices
- Dynamic unavailability of distinct devices

An additional usage of load generators in CGROUPS sandbox in combination with dynamic adjustment of parameters for each individual resource extends the scope. A graphical illustration of such a composition is shown in Figure 2. Further fault classes, as listed below, are introduced:

- CPU over load

 ISBN: 978-1-61208-212-7 

- Memory leaks

The review of related work identified most SWIFI approaches focus on processor faults some additional solutions support memory or I/O driver fault injection. Most of these techniques require an installation of special drivers others affect run time behavior of the operation system. Some methods depend on low-level changes in the operating system or modify the SuT during run time. Additionally, a few are based on low-level operating system functions which necessitate running an application in a special operational mode like trace-mode. The presented approach is portable to any LINUX system in more detail most distributions directly contain the required components. Thus, the architecture of an underlying computer system does not derogate the application of this approach.

### A. Implementation

For evaluation purpose the above presented method was implemented in a LINUX environment. In respect of API library LIBCG which works close to the kernel it was done in C/C++ language. Additionally a dynamic load generation and sandbox setup is supported by predefined profiles. These profiles are parsed by using BISON parser generator. The parsing results in an action list which triggers each individual resource object. The profile also includes all necessary information for automatically generating a CGROUP during application runtime. This group builds the sandbox for all execute actions.

### B. Action trigger

The initiating of all actions in the implementation is time triggered. This includes the dynamic adaption of CGROUP parameters as well as the activation, setup and adjustment of load generators. An exceptional case form consumed resources by load generators. These resources are predefined but the generated load depends on environmental conditions, such that the consumed amount does not exceed the available resources. The adjustment of sandbox parameters during runtime is not restricted or controlled in any way. Thus a generation of limiting cases is feasible

### C. Initial experiments

A set of initial experiments was done on a laptop and two servers with different hardware configurations. The spectrum from a dual-core processor up to a server with four HEXA-core processors was covered. Memory on these machines spanned range from 3GB to 16GB RAMS. The experiments were based on Ubuntu and Debian LINUX distributions.

## V. Conclusion

The paper presents an ongoing research for a new way to emulate resource faults in computer systems. A comparison of the concept with existing techniques in the area of SWIFI was done. To the best of our knowledge, such a kind of failure scenario was not presented before. The method runs without affecting operational behavior of the host operating system and does not necessitate the installation of special customized drivers. Furthermore, no low-level changes in the operating system or running an application in a special operational mode like trace-mode are required. The sandboxing of resources leads to an isolation of consumed and available resources. This enables testing under realistic conditions without interrupting normal operational behavior of the host system.

## Acknowledgment

I want to thank The Klaus Tschira Foundation gGmbH and Prof. Dr.-Ing. Dr. h.c. Andreas Reuter for funding and supporting this work.

## References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” University of Maryland / Institute for Systems Research, Tech. Rep., 2004.

[2] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, “Fault injection techniques and tools,” *Computer*, vol. 4, pp. 75–82, 1997.

[3] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, “Fault injection for dependability validation: A methodology and some applications,” *IEEE Trans. Softw. Eng.*, vol. 16, pp. 166–182, February 1990. [Online]. Available: http://dx.doi.org/10.1109/32.44380

[4] P. Tröger, F. Salfner, and S. Tschirpke, “Software-implemented fault injection at firmware level,” in *Proceedings of the 2010 Third International Conference on Dependability*, ser. DEPEND ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 13–16. [Online]. Available: http://dx.doi.org/10.1109/DEPEND.2010.10

[5] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, “Ferrari: A flexible software-based fault and error injection system,” *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 248 – 260, 1995.

[6] T. K. Tsai and R. K. Iyer, “Measuring fault tolerance with the ftape fault injection tool,” in *Proceedings of the 8th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation: Quantitative Evaluation of Computing and Communication Systems*. London, UK: Springer-Verlag, 1995, pp. 26–40. [Online]. Available: http://dl.acm.org/citation.cfm?id=648080.746851

[7] J. a. Carreira, H. Madeira, and J. a. G. Silva, “Xception: A technique for the experimental evaluation of dependability in modern computers,” *IEEE Trans. Softw. Eng.*, vol. 24, pp. 125–136, February 1998. [Online]. Available: http://dx.doi.org/10.1109/32.666826

[8] B. Singh and V. Srinivasan, “Containers: Challenges with the memory resource controller and its performance,” *Ottawa Linux Symposium*, vol. 2, pp. 209–222, 2007.

 ISBN: 978-1-61208-212-7 

# Design of Dependable Systems: An Overview of Analysis and Verification Approaches

Jose Ignacio Aizpurua, Eñaut Muxika
*Department of Signal Theory and Communications*
*University of Mondragon*
*Spain*
{*jiaizpurua,emuxika*}*@mondragon.edu*

***Abstract*—Designing a dependable system successfully is a challenging issue that is an ongoing research subject in the literature. Different approaches have been adopted in order to identify, analyse and verify the dependability of a system design. This process is far from obvious and often hampered due to the limitations of the classical dependability analysis techniques and verification approaches. This paper provides an overview of analysis approaches grouped by limitations. The principal points for the characterization of the considered approaches are the capability to handle notions of time, component-wise failure propagations and the use of architectural languages with the aim to extract analysis models from design models. Finally, verification approaches are partially reviewed.**

***Keywords-Dependability design; Dependability Analysis; Dependability Verification; Model-Based Analysis.***

## I. INTRODUCTION

The goal of this paper is to provide a list of sources to those readers who are not familiar to the field of model-based design of dependable systems. Our goal is not to exhaustively evaluate the specific features of these approaches, but to aggregate a comprehensive list of works grouped by their main characteristics.

In computing systems, dependability is defined as "ability of a system to deliver a service that can be justifiably trusted" [1]. Such trustworthiness is based on the assurance of dependability requirements. These requirements are defined through dependability attributes: *Reliability*, *Availability*, *Maintainability*, *Safety (RAMS)*, *confidentiality* and *integrity*. The scope of this overview focuses on RAMS attributes. Consequently, security aspects (confidentiality and integrity) are not addressed.

*Reliability* is the ability of an item to perform a required function under given conditions for a stated period of time [2]. *Maintainability* is the ability to undergo repairs and modifications to restore to a state in which the system can perform its required actions. *Availability* is the readiness for correct service and *safety* is the absence of catastrophic consequences on the user(s) and the environment.

This survey concentrates on three main phases: dependability analysis, system design and verification. Despite being aware of the relevance of software code for system dependability in each of these phases, we will consider software code as a black box component to limit the extension of this paper (interested readers refer to [3] [4]).

Dependability analysis techniques can be organised by looking at how different system failures are characterized with its corresponding underlying formalisms. On one hand, *event-based* approaches reflect the system functionality and failure behaviour through combination of events. This analysis results in either Fault Tree (FT) like [5] or Failure Mode and Effect Analysis (FMEA) like [6] structures, which emphasizes the reliability and safety attributes. On the other hand, *state-based* approaches map the analysis models into a state-based formalism (e.g., Stochastic Petri Nets (SPN)). Those approaches analyse system changes with respect to time and centre on reliability and availability attributes.

Fault injection and model-checking [7] approaches are mainly adopted for model-based analysis and verification of design decisions. Principally, they are aimed at checking and evaluating dependability requirements using nominal and failure behaviour models. This overview addresses the analysis and verification of system properties using these approaches. There is also another class of verification approaches, which try to ensure the validity of the system by design [8] (i.e., formal verification).

The remainder of this paper is organized as follows: Section II classifies dependability analysis techniques based on the limitations of classical techniques. Section III studies how to adopt these approaches when designing a dependable system. Section IV discusses the characteristics of verification approaches when designing a dependable system. Section V outlines an abstract hybrid design process based on the reviewed analysis, design and verification approaches. Finally, Section VI draws conclusions remarking different challenges for designing dependable systems. Due to space limitations, acronyms are used throughout the work. Interested readers can refer to listed references.

## II. REVIEW AND CLASSIFICATION OF DEPENDABILITY ANALYSIS TECHNIQUES

Event-based approaches analyse the failure behaviour of the system by investigating the logic succession of faults.

 ISBN: 978-1-61208-212-7

They identify an event sequence leading to equipment or function failure. Differences are mainly based on representation and analysis structures. Two of the most widely used techniques are: FT Analysis (FTA) [5] and FMEA [6].

Both techniques focus on the identification of events that jeopardize the objectives of the design. However, their logical deductive/inductive orientation (from known effects/causes towards unknown causes/effects respectively for FTA/FMEA) and initial assumptions are different. They are not orthogonal techniques, indeed they are complementary and in some cases they overlap. The extended usage of these approaches for dependability related tasks have lead to the identification of the main limitations. Subsequently, there has been a long list of works aimed at overcoming them:

- L1: FMEA and FTA are static representations of the system, neither time information nor sequence dependencies are taken into account [9].
- L2: Orientation of FTA and FMEA concentrate on the analysis of failure chain information. Consequently, their hierarchy reflects failure influences without considering system functional architecture information [10].
- L3: FMEA and FTA depend on the analyst's skill to reflect the aspects of interest. Failure modes (FM) and undesired events must be foreseen, resulting in a process highly dependent on analyst's knowledge of the system [11].
- L4: Manageability and legibility of FTA and FMEA models is hampered when analysing complex systems. Model size, lack of resources to handle interrelated failures and repeated events, in conjunction with few reusability means, are its main impediments [10] [12].

L1 refers to the capability of the technique to handle temporal notions. This is of paramount importance when analysing fault tolerant systems. L2 emphasizes the interdisciplinary work between dependability analysis and architectural design. Joining both procedures helps obtaining a design, which meets dependability requirements consistently. L3 entails a trade-off solution between the time consuming analysis resulted from understanding the failure behaviour of the system and the acquired experience. A substantial body of works have been oriented towards the automatic generation of analysis models from design models (refer to groups 3, 5 in Table I) addressing limitations L2 and L3. These approaches promote the reuse of design models showing the effects of design changes in the analysis results. However, note that the correctness of the analysis lies in the accuracy of the failure annotations. Finally, L4 underlines the capability of the model to handle the component-wise nature of embedded systems. This permits obtaining a model that better adheres to the real problem and avoids confusing results.

Many authors have developed new alternatives or extended existing ones. Three groups are identified in order to gather the approaches and limitations strategically. Firstly, L1 is addressed in the subsection *dynamic solutions for static-logic approaches*. Secondly, L2 and L4 are covered in *compositional failure analysis approaches*. Finally, specifically focusing on L3 and generally addressing the remainder of limitations *model-based transformational approaches* are studied. Note that some approaches cannot be limited to a specific group, hence they are classified accordingly to its main contribution.

### A. Dynamic Solutions for Static-Logic Approaches

The limitation concerning the lack of time information has been addressed by several authors to deal with system *dynamics* such as redundancy or repair strategies.

Dugan et al. [9] paved the way to cope with configuration changing analysis using FTs by defining Dynamic Fault Tree (DFT) methodology. New gates were introduced accounting for event ordered situations like common cause failures and redundancy strategies. In [13], temporal notions and FT models were integrated in order to handle the timed behaviour of the system. The model reflects how modular FT models are switched through discrete points in time taking into account time dependant basic events.

Other alternatives to analyse DFT models are based on Monte Carlo simulations (MCS) by specifying temporal failure and repair behaviours of components through state-time diagrams [14]. In [15], an approach based on Simulink [16] for DFT modelling and reliability analysis is presented. The technique integrates MCS and FT methodologies resulting in a intuitive model-based simulating environment.

Following the way of DFTs, an approach emerged based on Reliability Block Diagrams (RBD) [2]. RBD is focused on the analysis of the success of the system, instead of the failure analysis orientation of FTs. Dynamic RBDs (DRBDs) [17] models failures and repairs of components based on their dependencies and state machines.

Lopatkin et al. [18] utilise FMEA models for system formal specifications. The approach defines generic patterns establishing direct correspondence between FMEA and state-based Event-B [19] formalism. Consideration of error detection and recovery patterns lead to analysing and verifying whether safety properties are preserved in the presence of faults. Utilization of these patterns, allows tracing from static FMEA considerations towards system *dynamics*.

Progression in the conjoint use of event and state formalisms is reflected with Boolean logic Driven Markov Processes (BDMP) [20]. BDMP employs static FT as a structure function of the system and associates Markov processes to each leaf of the tree. Similarly, State-Event Fault Tree (SEFT) [21] formalism combines elements from FT with both Statecharts and Markov chains, increasing the expressiveness of the model. SEFT deals with functional and failure behaviour, accounts for repeated states and events and

 ISBN: 978-1-61208-212-7

allows straightforward transformations of SEFT models into Deterministic and Stochastic Petri Net (DSPN) models for state-based analysis.

The compositional and transformational features of the SEFT approach, provide an adequate abstraction of the system structure and behaviour. As far as our knowledge, there is no available tool for the evaluation and transformation of SEFT models. Developing a model-based tool which extracts DSPN models from SEFT models, would create an adequate environment for constituting a sound approach for manageable and consistent dependability analyses.

### *B. Compositional Failure Propagation Analysis Approaches*

The common factors for Compositional Failure Propagation (CFP) approaches are: the characterization of the system architectures by design components; the annotation of the failure behaviour of each of them; and the system failure analysis based on inter-components influences. Conceptually they all are very similar: the main objective of CFP approaches is to avoid unexpected consequences resulting from the failure generation, propagation and transformation of components.

Generally, CFP approaches characterise the system as component-wise developed FT-like models linked with a causality chain. System architectural specifications and subsequent dependability analyses of CFP approaches rely on a hierarchical system model. This model comprises components composed from subcomponents specifying system structure and/or behaviour. CFP approaches analyse the system failure behaviour through characterizations of individual components, which lead to achieving a manageable failure analysis procedure. Failure Propagation and Transformation Notation (FPTN) [22], Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [23] and Component Fault Tree (CFT) [10] are the principal CFP approaches. Their main difference is in the failure annotations of components, which specify incoming, outgoing and internal failures to each component. In order to annotate the logical combinations of these failures, FPTN uses logical equations, HiP-HOPS makes annotations using Interface Focused FMEA (IF-FMEA) tables and CFT associates to each component individual FTs. Subsequently, the connections between system components determines the *failure flow* of the system, linking related failure annotations of each component.

Concerning the different contributions of CFP approaches, FPTN first addressed the integration of system-level deductive FTA (from known effects to unknown causes) with component-level inductive FMEA (from known causes to unknown effects). HiP-HOPS integrates design and dependability analysis concepts within a hierarchical system model. However, instead of exclusively linking functional components with their failure propagations like in FPTN, first the hierarchical system model is specified and then compositional failure annotations are added to each component by means of IF-FMEA annotations. These annotations describe the failure propagation behaviour of the component in terms of outgoing failures specified as logical combinations of incoming and internal failures. Subsequently, a FT synthesis algorithm analyses the propagation of failures between connected components. Traversing the hierarchical system model, while parsing systematically the IF-FMEA annotations of its constituent components, allows the extraction of the system FT and FMEA models. CFT works in a slightly different way, it aims at linking FTs of the components with the architecture design. The component FTs can be combined and reused to systematically obtain the FT for any failure without having to create and annotate a FT for each failure.

They all have been extended to cope with occurrences of temporal events. Temporal extensions for FPTN [24] and HiP-HOPS [25] have been influenced by the DFT methodology. Focusing on non-repairable systems, the order of events is examined in order to identify specific sequence of events leading to failures. Integration of CFT concepts with state-based techniques resulted in SEFT formalism, which is able to handle availability and maintainability properties of repairable systems.

Transformation of CFP approaches into dependability analysis formalisms is an ongoing research subject (see Subsection II-C). HiP-HOPS extracts FTA and FMEA models thanks to its underlying logic and SEFT applies a translation algorithm to generate DSPN models.

Other interesting extensions include mechanisms to automate and reuse analysis concepts. Failure Propagation and Transformation Calculus (FPTC) [26] approach introduces notations to indicate nominal behaviour within FPTN models and concepts to generalise and manage FPTN equations. Moreover, an algorithm is implemented handling the general inability of CFP approaches to cope with cyclic dependencies of feedback structures. In [27], general failure logic annotation patterns were defined for HiP-HOPS. Similarly, the CFP approach presented in [28], emphasizes the reuse of failure propagation properties specified at the port level of components. These specifications centre on the physical properties of different types of flows, which allow reusing failure behaviour patterns for functional architectures.

The evolution of CFP approaches focus on reusability, automation and transformation properties. Since the annotations of components failure behaviour depend upon designers subjective considerations, reusing failure annotations leads to reducing the error proneness. Based on the fact that different dependability analyses have to be performed when designing a dependable system, definition of a unique consistent model covering all analyses would benefit these approaches. This is why recent publications in this field centre on integrating existing approaches (cf. Subsection II-C and Section IV).

 ISBN: 978-1-61208-212-7

### C. Model-Based Transformational Approaches

The main aim of the transformational models is to construct dependability analysis models (semi-)automatically. The process starts from a compositional design description using computer science modelling techniques. The failure behaviour is specified either by extending explicitly the design model or developing a separate model, which is allocated to the design model. Transformation rules and algorithms extract dependability analysis models from it.

These approaches lead to adopting a trade-off decision between dependability design and analysis processes. On one hand, the automation and reuse of analysis techniques in a manageable way makes it a worthwhile approach for design purposes. The impact of design changes on dependability attributes are straightforwardly analysed. On the other hand, from purist's point of view of classical analysis techniques, the automation process removes the ability of these techniques to identify and analyse hazards or malfunctions in a comprehensive and structured way.

Architectural description languages (ADLs) provide an adequate abstraction to overcome the limitations. Simulink [16], AADL [29] and UML [30] have been used for both architectural specification and failure behaviour specification. UML is a widely used modelling language, which has been extended for dependability analyses following model driven architecture (MDA) concepts. Namely, profiles [31] allow extending and customizing modelling mechanisms to the dependability domain.

Lately, a wide variety of independently developed extensions and profiles have come up for dependability analysis [32]. However, some generally applicable metamodel is lacking. In an effort to provide a consistent profile CHESS ML [33] emerged. Its high-level specifications are transformed into Intermediate Dependability Model (IDM) in order to facilitate transformations. CHESS ML development is currently ongoing and seems to provide all necessary mechanisms to model dependable systems and extract either event-based (FMECA, FPTC) or state-based (SPN) analysis models.

CFP approaches have been shifted towards the transformational paradigm. The toolset for FPTC approach [26] relies on a generic metamodel in order to support transformations from SysML and AADL models. In [34], a metamodel is developed so as to extract CFT models from functional architecture models specified in UML. This process permits the generation of reusable CFT models consistent with the design model. In the same line, integration of HiP-HOPS model with EAST-ADL2 automotive UML profile is presented in [35]. Translations from high-level ADLs to well established CFP analysis techniques, enable an early dependability analysis and allow undertaking timely design decisions.

Architecture Analysis and Design Language (AADL) captures the system architectural model in terms of components and their interactions describing functional, mapping and timing properties among others. The core language can be extended to meet specific requirements with annex libraries. Behaviour and error model annexes are provided with the tool. The error annex links system architecture components to their failure behaviour specification making possible the analysis of the dependability attributes of the system. It has been used for both state-based [36] and event-based [37] analysis.

AltaRica [38] is a dependability language, which enables describing the behaviour of systems when faults occur. The model is composed of several components linked together representing an automaton of all possible behaviour scenarios, including those cases when reconfigurations occur due to the occurrence of a failure [39]. It is possible to process such models by other tools for MC or generation of FTs [40]. Transformations from AADL to AltaRica are presented in [41], based on MDA concepts so as to perform dependability analyses and avoid inconsistencies while working with different formalisms.

In [42], a method for RAMS analysis is defined centred on SysML [43] modelling language from where a FMEA model is deduced. SysML diagrams define a functional model connected to a dysfunctional database enabling the identification of FMs. This database contains the link between system architecture and failure behaviour giving the key for FMEA extraction. Further, the methodology for dependability assessment is extended using AltaRica, AADL and Simulink models. They address reliability and timing analysis and simulation of the effects of faults respectively.

Definition of a model for the extraction of all necessary formalisms for dependability analysis is the common goal for the aforementioned works. Interconnections between different formalisms in order to take advantage of the strengths of each ADL, allow analysing dependability properties accurately. AltaRica and AADL cover adequately the analysis of reliability, availability and maintainability attributes. Extraction of the main CFP approaches from ADLs should help to analyse comprehensively system safety properties. Moreover, Simulink model simulations allow evaluating the effects of failure and repair events in the system. Thereby, integrations between language specific models like in [42] helps evaluating accurately all dependability aspects of a system.

### D. Classification of Techniques

In order to classify the covered approaches, Table I groups them taking into account limitations specified in Section II.

Approaches gathered within the group 5 contain all necessary features in order to analyse dynamic systems consistently and in a manageable way. Compositional failure annotation, dynamic behaviour and automatic extraction of analysis models are the key features addressed by these

 ISBN: 978-1-61208-212-7

Table I
SUMMARY OF LIMITATIONS OVERCOME BY APPROACHES

| Group | Approach | Limitations |
|---|---|---|
| 1 | [9] [13] [14] [20] | L1 |
| 2 | [10] [22] [26] | L2, L4 |
| 3 | [23] [28] [37] | L2, L3, L4 |
| 4 | [17] [24] [39] | L1, L2, L4 |
| 5 | [21] [18] [25] [33] [36] [42] | L1, L2, L3, L4 |

approaches. Utilization of failure annotation patterns promote flexibility and reuse and consequently, reduce the error proneness. Nevertheless, as noted in [44], characterization of the failure behaviour of components depends on the component context, which conditions compositional and reuse properties. Moreover, automatic generation of the analysis model does not completely alleviate the dependency on the knowledge of the analyst. However, it lets managing and specifying the failure behaviour in a clear and consistent way.

## III. DEPENDABLE DESIGN: TRADE-OFF BETWEEN DEPENDABILITY AND COST

Generally, dependability design decisions and objectives are related to trade-off decisions between system dependability attributes and cost. Dependability requirements often conflict with one another e.g., safety-availability compromise when a faults leads the system to a safe shutdown in order to prevent it from propagating. The time at which design decisions are taken, determines the cost that the design process can incur.

Designing a dependable system within considered requirements requires a process to match and tune combination of architectural components so as to find an optimal solution satisfying design constraints. For the sake of analysing the applicability of the aforementioned analysis techniques, three demonstrative works are chosen. Their underlying structure is illustrated in Figure 1.

Dependable design methodologies are proposed by Benard et al. [45] and Clarhaut et al. [46]. The former focuses on quantification and comparison of RAMS properties of alternative components. The latter overcomes static-logic limitations by integrating temporal functions. Their design methodology is based on the *operational model*, which aims at mapping the *functional model* onto a *compatible physical model* (cf. Figure 1).

The functional model is developed in a top-down hierarchical manner tracing from system level functions up to lower level functions. At the lowest level, physical components are linked with corresponding functions. Dependability considerations lead to characterizing hardware components through failure modes (*FM*) and *redundancy structures*. The way in which *dependability analysis* is performed differ both approaches. While the former uses MCS to analyse the

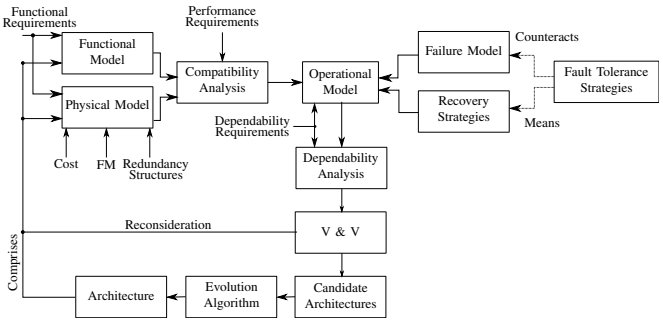


Figure 1. Abstract Design Process (Adapted from [45])

impact of allocations of functions into physical components; the latter focuses on identifying component-wise temporal failure contributions to the system-level undesired event.

In [47], HiP-HOPS approach is extended with *recovery strategies*. These capabilities are formally represented using patterns. They characterize the potential to detect, mitigate and block affecting component failures identified in the *failure model*. Dependability analysis is performed by means of FTA and FMEA.

Trade-off analysis between dependability and cost determines optimal *architectures*. In [47], fault tolerant configurations are introduced without violating user constraints and an *evolutionary optimization algorithm* is used to converge through dependability and cost requirements. Similarly [46] identifies set of optimal *candidate architectures* by minimizing failure contributions and cost of necessary components to accomplish system functions.

## IV. DEPENDABLE DESIGN VERIFICATION: FAULT INJECTION APPROACHES

Fault Injection (FI) techniques focus on evaluating system behaviour in the presence of faults according to target dependability properties. The outcome of this process may lead to considering design changes. However, changes adopted late in the design process are costly. This is why we focus on FI approaches adopted at the preliminary design phase. This process is based on the analyst's knowledge to reason about the functional and failure behaviour of the system. As a result, the effectiveness of fault detection, isolation, recovery and reconfiguration strategies are evaluated. Timely evaluation of these properties provides a valuable feedback for design purposes. However, difficulties arise from the accuracy of the system behaviour, which requires an accurate knowledge of the system.

Instead of focusing on purely verification oriented FI approaches, we address *integrative verification* approaches. These works result from the integration of design, analysis and verification tasks. Covered approaches aim at combining dependability analysis techniques examined within the group 5 (cf. Table I) with FI approaches. They express system

 ISBN: 978-1-61208-212-7

behaviour using a compositional model, which gathers nominal, failure and recovery behaviours. Integrating approaches using model transformations, allows using a single design model for dependability and verification analyses.

The system *design model* takes into account functional and failure behaviour of components. Temporal logic languages are used to define system *requirements*. They describe how the truth values of assertions change over time, either qualitatively (Computation Tree Logic (CTL), Linear Time Logic (LTL)) or quantitatively (Continuous Stochastic Logic (CSL), probabilistic CTL (PCTL)). Model-checking (MC) engine assesses whether such requirements are met or not by the design model, using a *analysis model*. To do so, it is necessary to transform the design model into the analysis model. When the analysis model fails to meet these requirements, its effects are deduced automatically identifying the paths that violate the conditions (counter-examples (CEs)) [7]. The logical orientation of this analysis process results in FMEA-like cause-effect analysis.

There are some limitations hampering the analysis and interpretation of these approaches. Representation structures of the results, state-explosion problems, technical specification difficulties, qualitative nature of MC analysis and model inconsistencies are some challenges to be addressed.

The COMPASS project [48] addresses these limitations based on SLIM (System-Level Integrated Modeling) language [49]. The semantics of SLIM cover the nominal and error behaviour of AADL. The complete specification of SLIM consists of a nominal model, a failure model and a description of the effects of failures in the nominal model (*extended model*). Due to its underlying formal semantics, various types of analyses are possible: validation of functional, failure, and extended models via simulation and MC; dependability analysis and performance evaluation; diagnosability analysis; and evaluation of the effectiveness of fault detection, isolation and recovery strategies.

Similarly, Güdemann and Ortmeier [50] proposed an intermediate (IM) tool-independent model called Safety Analysis Modelling Language (SAML). SAML describes a finite state automata, which is used to characterise the extended system model. This model specifies the nominal behaviour, failure occurrences, its effects and the physical behaviour of the surrounding environment. From this single model, quantitative and qualitative MC analyses are performed. The former identifies minimal critical sets using CEs to indicate time-ordered combinations of failures causing the system hazard. The latter calculates per-demand and per-time failure probabilities.

TOPCASED project [51] aims at developing critical embedded systems including hardware and software. TOPCASED integrates ADLs and formal methods. The approach transforms high-level ADL models (SysML, UML and AADL) into an IM model specified in Fiacre language [52]. Fiacre specifies behavioural and timing aspects of high-level models making use of timed Petri nets primitives. Subsequent transformations of the IM model into MC tools (TINA and CADP), make possible the formal verification and simulation of the specified requirements. TINA [53] analyse requirements specified in the state variant of LTL proposition logic (State/Event LTL (SELTL)) focusing on timeliness properties. CADP [54] transforms Fiacre models into LOTOS programs, which are handled by its underlying tools for validation via MC and simulation.

Albeit these approaches provide a means to extract classical dependability models from high-level models, none of them focus on integrating existing CFP approaches. There are some incipient works linking CFP and verification approaches. They are influenced by HiP-HOPS [55] and FPTC [56]. Both approaches address the integration of qualitative design models with quantitative analysis via probabilistic MC. These approaches in particular and CFPs in general, provide useful resources when characterizing the failure behaviour of systems. The pros and cons of the covered works are summarized in the Table II.

The addressed works integrate well known tools and formalisms. However, integration of analysis and verification approaches when designing a dependable system is an ongoing research subject. There is an increasing interest in reusing and generalizing CFP approaches (e.g., transformation of CFP approaches into metamodels [26] [34] [35] and integration of CFP and verification approaches [55] [56]).

## V. Hybrid Design Process

The goal of this section is not to provide a new design approach. Our aim is to make use of the reviewed analysis, design and verification approaches so as to outline a consistent and reusable model-based design process. This process emerges from the structure of the integrative verification approaches (cf. Section IV).

The separation of dependability analysis and verification tasks may lead to hampering the system design since results identified from either task need to be reconsidered during the design process (cf. Section III). On one hand, dependability analyses characterized by transformational approaches (cf. Section II-C), allow tracing from design considerations towards dependability analysis models. These approaches evaluate the dynamic system behaviour, as well as the effect of particular component failure occurrences at system level. On the other hand, purely verification oriented approaches mainly focus on the verification of the adequacy of the design model with respect to RAMS requirements. This is why we centre on covering integrative verification approaches.

When matching and tuning design components so as to find optimal design solutions satisfying design constraints, possible inconsistencies may arise due to the independent considerations of these approaches. This is why we should focus on outlining a model-based hybrid design process, which unifies design, analysis and verification tasks. This

 ISBN: 978-1-61208-212-7

Table II
SUMMARY OF FAULT INJECTION APPROACHES

| *Works* | *Design Model* | *Analysis Model (*Auto)* | *Reqs. (*Auto)* | *Results* | *Specific Features* | *Future works* |
|---|---|---|---|---|---|---|
| [49] | SLIM | NuSMV*, MRMC* | CTL*, LTL*, CSL* | DFT, FMEA, Prob. Calc. | Req. patterns; Integrated verification and dependability and performance analyses of extended models. | Manual extension of the nominal model; Redundant FTA-FMEA results; State-explosion. |
| [50] | SAML | NuSMV*, PRISM*, MRMC* | CTL, PCTL | Time-Ord. CE, Prob. Calc. | Combination of qualitative and quantitative analyses on the same model. | Manual extension of the nominal model; Transf. ADLs (Simulink, Scade) into SAML; Req. patterns. |
| [52] | Fiacre | TINA, CADP | LTL, SELTL | Timing, Prob. Calc. | Req. patterns; Integrated design, analysis and verification approaches. | State Explosion; Back annotation of results. |
| [55] | Simulink | PRISM* | CSL | Prob. Calc. | Systematic generation of analysis models from CFP design models. | No CE; State-explosion; Dynamic behaviour. |
| [56] | FPTC | PRISM | CSL | CE | Integration of CFP approach and prob. model checking. | Fail Prone Manual transformation; Translate CE to design model. |

process relies on initial system requirements, models, transformations and reuse of designer's considerations and results extracted from analysis and verification tasks (cf. Figure 2).

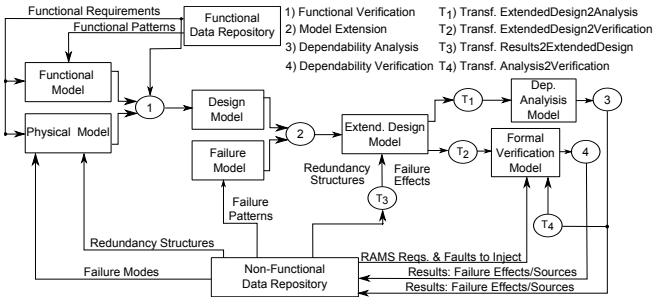


Figure 2. Hybrid Design Process

This design process starts from initial functional and physical considerations. *Functional verification* analysis evaluates the adequacy of the allocation of the *functional model* into the *physical model* according to functional requirements. The outcome of this process allows considering the verified *design model* (operational model, cf. Figure 1). Subsequently, this model is extended with the *failure model* accounting for failure occurrences of the considered model. Failure patterns aid in the construction of the failure model allowing the reuse non-functional considerations. Further, the effects of the considered failures and recovery strategies are annotated in the *extended design model* in order to counteract failure occurrences and its effects. With the aim to carry out *dependability analysis* and *formal verification* evaluations of the extended design model, twofold transformations need to be performed. The means to perform these transformations have been presented in Subsection II-C and Section IV respectively. Transformations of these models make the evaluation of the adequacy of the extended design model respect to RAMS requirements possible. Dependability analysis and verification tasks enable finding further failure effects and failure sources (apart from occurrence probabilities) either by CEs or dependability specific models. These results need to be transformed in order to reconsider for design and analysis purposes. For the sake of reusing and refining the design process, data repositories have been considered consisting of annotation patterns for requirements and models (both functional and non-functional) and reusable recovery strategies.

On one hand, the outlined design approach enables benefiting from consistent design considerations. Moreover, data repositories allow the reuse of designer's considerations as well as analysis results. Furthermore, user-friendly means make the annotation processes more evident. On the other hand, the automation of the extraction of dependability models hides information about the failure behaviour. Additionally, the flexibility of the approach depends on the system context, which would determine the reusability of functional and non-functional considerations.

## VI. CONCLUSION AND FUTURE WORK

Designing a dependable system, poses a wide variety of challenges on all its phases. This paper groups different approaches in order to identify and classify them.

The listed limitations guided the evolution of the analysis techniques towards Compositional Failure Propagation (CFP) and transformational approaches. Automatic extraction of analysis models from design models is an ongoing research field, which leads to achieving consistency between design and analysis models.

However, this is not the cure-all remedy, which alleviates analysts from identifying and analysing failure behaviours, but helps obtaining a manageable analysis compared to the difficult and laborious traditional process. User friendly resources, such as design components or failure annotation libraries, enable the reuse of nominal and failure models.

When designing a new system, special care should be taken, since reuse properties depend on the system context.

 ISBN: 978-1-61208-212-7 

The reuse of failure annotations in the design process, eases the architectural iterative refinement process. This makes possible the analysis of different implementations using the same component failure models.

For verification purposes, fault injection (FI) approaches have been studied. Since the adoption of FI approaches is made late in the traditional design process, we have considered integrative works. Their main objective is to address consistently dependability analysis, design and verification tasks at the preliminary design phase. An early integration of these tasks would add value to the dependable design process. There are many challenging tasks to address when constructing an end-to-end dependable design methodology. Integration of the CFP approaches within this methodology or validation of the correctness of the faults to be injected are some of the subjects to be addressed.

Therefore, we hypothesize that instead of developing independent approaches to identify, analyse and verify dependability requirements, future directions will focus on integrating different approaches. This process requires tracing verification results to the initial dependable design model and vice versa. Consequently, accounting for these considerations, we have sketched an abstract integrative design process. The integration of the approaches should allow undertaking timely design decisions by reducing costs and manual failure-prone annotations. Additionally, it will alleviate the need to clutter a model with redundant information. In this field, challenging work remains to do sharing information between existing approaches so as to take advantage of complementary strengths of different approaches.

## REFERENCES


[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, pp. 11–33, January 2004.

[2] M. Rausand and A. Høyland, *System Reliability Theory: Models, Statistical Methods and Applications Second Edition*. Wiley-Interscience, 2003.

[3] A. Arora and S. Kulkarni, "Component based design of multitolerant systems," *IEEE Trans. on Sw. Eng.*, vol. 24, no. 1, pp. 63–78, 1998.

[4] M. Hiller, A. Jhumka, and N. Suri, "An approach for analysing the propagation of data errors in software," in *Proc. of DSN'01*, 2001, pp. 161–170.

[5] W. Vesely, J. Dugan, J. Fragola, Minarick, and J. Railsback, "Fault Tree Handbook with Aerospace Applications," NASA, Handbook, 2002.

[6] US Department of Defense, *Procedures for Performing, a Failure Mode, Effects, and Criticality Analysis (MIL-STD-1629A)*. Whasington, DC, 1980.

[7] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[8] Y. Prokhorova, L. Laibinis, E. Troubitsyna, K. Varpaaniemi, and T. Latvala, "Derivation and formal verification of a mode logic for layered control systems," in *Proc. of APSEC'11*, 2011, pp. 49–56.

[9] J. Dugan, S. Bavuso, and M. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Trans. on Reliability*, vol. 41, no. 3, pp. 363–377, 1992.

[10] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proc. of SCS'03*, 2003, pp. 37–46.

[11] A. Galloway, J. McDermid, J. Murdoch, and D. Pumfrey, "Automation of system safety analysis: Possibilities and pitfalls," in *Proc. of ISSC'02*, 2002.

[12] C. Price and N. Taylor, "Automated multiple failure FMEA," *Reliability Eng. & System Safety*, vol. 76, pp. 1–10, 2002.

[13] M. Ĉepin and B. Mavko, "A dynamic fault tree," *Reliability Eng. & System Safety*, vol. 75, no. 1, pp. 83–91, 2002.

[14] Rao, K. Durga, V. Gopika, V. V. S. Sanyasi Rao, H. S. Kushwaha, A. K. Verma, and A. Srividya, "Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment," *Reliability Eng. and System Safety*, vol. 94, no. 4, pp. 872–883, 2009.

[15] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, and N. Trapani, "MatCarloRe: An integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10 334–10 342, 2012.

[16] "MathWorks," http://www.mathworks.com; Last access: 2012/06/13.

[17] S. Distefano and A. Puliafito, "Dynamic reliability block diagrams vs dynamic fault trees," *In Proc. of RAMS'07*, vol. 8, pp. 71–76, 2007.

[18] I. Lopatkin, A. Iliasov, A. Romanovsky, Y. Prokhorova, and E. Troubitsyna, "Patterns for representing FMEA in formal specification of control systems," in *Proc. HASE'11*, 2011, pp. 146–151.

[19] "Event-B and the Rodin platform," http://www.event-b.org; Last access: 2012/06/13.

[20] M. Bouissou, "A generalization of Dynamic Fault Trees through Boolean logic Driven Markov Processes (BDMP)," in *Proc. of ESREL'07*, vol. 2, 2007, pp. 1051–1058.

[21] B. Kaiser, C. Gramlich, and M. Forster, "State/event fault trees a safety analysis model for software-controlled systems," *Reliability Eng. System Safety*, vol. 92, no. 11, pp. 1521–1537, 2007.

[22] P. Fenelon and J. A. McDermid, "An integrated tool set for software safety analysis," *J. Syst. Softw.*, vol. 21, pp. 279–290, 1993.

[23] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, and R. Lien, "Engineering failure analysis and design optimisation with HiP-HOPS," *Engineering Failure Analysis*, vol. 18, no. 2, pp. 590–608, 2011.


 ISBN: 978-1-61208-212-7 

[24] R. Niu, T. Tang, O. Lisagor, and J. McDermid, "Automatic safety analysis of networked control system based on failure propagation model," in *Proc. of ICVES'11*, 2011, pp. 53–58.

[25] M. Walker and Y. Papadopoulos, "Qualitative temporal analysis: Towards a full implementation of the fault tree handbook," *Control Eng. Practice*, vol. 17, no. 10, pp. 1115–1125, 2009.

[26] R. Paige, L. Rose, X. Ge, D. Kolovos, and P. Brooke, "FPTC: Automated safety analysis for Domain-Specific languages," in *MoDELS Workshops '08*, vol. 5421, 2008, pp. 229–242.

[27] I. Wolforth, M. Walker, L. Grunske, and Y. Papadopoulos, "Generalizable safety annotations for specification of failure patterns," *Softw. Pract. Exper.*, vol. 40, pp. 453–483, 2010.

[28] C. Priesterjahn, C. Sondermann-Wölke, M. Tichy, and C. Hölscher, "Component-based hazard analysis for mechatronic systems," in *Proc. of ISORCW'11*, 2011, pp. 80–87.

[29] P. Feiler and A. Rugina, "Dependability Modeling with the Architecture Analysis & Design Language (AADL)," Technical Note CMU/SEI-2007-TN-043, CMU Software Engineering Institute, 2007.

[30] "The Unified Modeling Language," http://www.uml.org/; Last access: 2012/06/13.

[31] L. Fuentes-Fernández and A. Vallecillo-Moreno, "An Introduction to UML Profiles," *UPGRADE*, vol. 5, no. 2, pp. 5–13, 2004.

[32] S. Bernardi, J. Merseguer, and D. Petriu, "Dependability modeling and analysis of software systems specified with UML," *ACM Computing Survey*, In Press.

[33] L. Montecchi, P. Lollini, and A. Bondavalli, "An intermediate dependability model for state-based dependability analysis," University of Florence, Dip. Sistemi Informatica, RCL group, Tech. Rep., 2011.

[34] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J. Schwinn, and M. Trapp, "Integration of component fault trees into the UML," in *MoDELS'10*, 2010, pp. 312–327.

[35] M. Biehl, C. DeJiu, and M. Törngren, "Integrating safety analysis into the model-based development toolchain of automotive embedded systems," in *Proc. of LCTES '10*. ACM, 2010, pp. 125–132.

[36] A. Rugina, K. Kanoun, and M. Kaâniche, "A system dependability modeling framework using AADL and GSPNs," in *Architecting dependable systems IV, LNCS*, vol. 4615. Springer, 2007, pp. 14–38.

[37] A. Joshi, S. Vestal, and P. Binns, "Automatic Generation of Static Fault Trees from AADL models," in *DNS Workshop on Architecting Dependable Systems*. Springer, 2007.

[38] A. Arnold, G. Point, A. Griffault, and A. Rauzy, "The AltaRica formalism for describing concurrent systems," *Fundamenta Informaticae*, vol. 40, no. 2-3, pp. 109–124, 1999.

[39] B. Romain, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge, "Experiments in model based safety analysis: Flight controls," in *DCDS'07*, 2007, pp. 43–48.

[40] P. Bieber, C. Castel, and C. Seguin, "Combination of fault tree analysis and model checking for safety assessment of complex system," in *Proc. of EDCC'02*, vol. 2485. Springer, 2002, pp. 624–628.

[41] K. Mokos, P. Katsaros, N. Bassiliades, V. Vassiliadis, and M. Perrotin, "Towards compositional safety analysis via semantic representation of component failure behaviour," in *Proc. of JCKBSE'08*. IOS Press, 2008, pp. 405–414.

[42] R. Cressent, V. Idasiak, F. Kratz, and P. David, "Mastering safety and reliability in a Model Based process," in *Proc. of RAMS'11*, 2011.

[43] "OMG Systems Modelling Language," http://www.omgsysml.org/; Last access: 2012/06/13.

[44] O. Lisagor, "Failure logic modelling: A pragmatic approach," Ph.D. dissertation, Department of Computer Science, The University of York, 2010.

[45] V. Benard, L. Cauffriez, and D. Renaux, "The Safe-SADT method for aiding designers to choose and improve dependable architectures for complex automated systems," *Reliability Eng. & System Safety*, vol. 93, no. 2, pp. 179–196, 2008.

[46] J. Clarhaut, S. Hayat, B. Conrard, and V. Cocquempot, "Optimal design of dependable control system architectures using temporal sequences of failures," *Ieee Transactions On Reliability*, vol. 58, no. 3, pp. 511–522, 2009.

[47] M. Adachi, Y. Papadopoulos, S. Sharvia, D. Parker, and T. Tohdo, "An approach to optimization of fault tolerant architectures using hip-hops," *Softw. Pract. Exp.*, 2011.

[48] "Correctness, Modelling and Performance of Aerospace Systems," http://compass.informatik.rwth-aachen.de; Last access: 2012/06/13.

[49] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Nguyen, T. Noll, and M. Roveri, "Safety, dependability and performance analysis of extended aadl models," *Computer Journal*, vol. 54, no. 5, pp. 754–775, 2011.

[50] M. Güdemann and F. Ortmeier, "Towards model-driven safety analysis," in *Proc. of DCDS 11*, 2011, pp. 53 – 58.

[51] "The Open-Source Toolkit for Critical Systems," http://www.topcased.org; Last access: 2012/06/13.

[52] B. Berthomieu, J.-P. Bodeveix, P. Farail, M. Filali, H. Garavel, P. Gaufillet, F. Lang, and F. Vernadat, "Fiacre: an intermediate language for model verification in the topcased environment," in *ERTS'08*, 2008.

[53] "TINA," http://projects.laas.fr/tina; Last access: 2012/06/13.

[54] "CADP," http://http://www.inrialpes.fr/vasy/cadp/; Last access: 2012/06/13.

[55] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and J. Buzzi, "Systematic model-based safety assessment via probabilistic model checking," in *ISoLA'10*. Springer, 2010, pp. 625–639.

[56] X. Ge, R. Paige, and J. McDermid, "Probabilistic failure propagation and transformation analysis," in *SAFECOMP'09*, 2009, vol. 5775, pp. 215–228.

 ISBN: 978-1-61208-212-7 

# Dependable Design: Trade-Off Between the Homogeneity and Heterogeneity of Functions and Resources

Jose Ignacio Aizpurua, Eñaut Muxika
*Department of Signal Theory and Communications*
*University of Mondragon*
*Spain*
{*jiaizpurua,emuxika*}*@mondragon.edu*

***Abstract*—Designing a dependable system by reducing costs is a challenging issue. Traditional design strategies replicate resources in order to improve fault-tolerant capabilities of the system, which leads to increasing the hardware cost. Originated from over-dimensioning decisions, we outline an approach to identify and gather inherent compatible resources of the system to accomplish equivalent functions. Inference of reconfiguration strategies from the inherent design redundancies of the system not only decreases the hardware cost, but also maintains, or even improves the dependability of the system.**

***Keywords*-Dependable design, distributed and reconfigurable systems, shared (quasi) redundancy.**

## I. Introduction

When designing a dependable system [1], redundancies improve system safety and availability. However, the aggregation of resources leads to higher costs and more failure sources. Consequently, reliability decreases. Therefore, a trade-off analysis between dependability and cost objectives is necessary to design a dependable system.

In order to perform a function within distributed networked control systems (NCSs), remote devices work in cooperation. A sensor performs a measurement function and sends it to a control algorithm through the network. The control algorithm acts in consequence and sends actuation commands to the remote actuator. Traditionally, sensors and actuators accomplish a single function, while processing units (PUs) handle multiple tasks. However, why not exploit sensor and actuator strengths to perform as many functions as they can?

Our design approach focus on NCSs operating under massively networked scenarios, where a lot of PUs and sensors are connected to a network for different purposes. E.g., a train where the resources of each car are replicated throughout the train cars or a building where room and floor resources are replicated.

In this paper, we propose a system design approach for the systematic identification of replaceable functions including those performed by sensors and actuators. To do so, the physical location is the key driver. Subsequent steps allow associating and deducing inherent design redundancies of the system. This approach allows improving specifically system availability and generally system dependability without additional hardware cost.

The remainder of this paper is organised as follows: Section II gives an overview of related research work. Section III describes the generic functional modelling approach. Section IV describes an overall reconfiguration process for designing a dependable system. Finally, Section V addresses limitations and future objectives of our research.

## II. Background Research

This section is divided into two subsections: Subsection II-A points out inspirations of our research and Subsection II-B identifies conceptually aligned works.

### A. Research Inspiration

A straightforward way to add redundancies to a system design is to explicitly replicate components. The objective of the added resources is to provide failover capabilities to a dedicated component failure. These replications are usually done using *passive* and *active redundancies*.

*Shared-redundancy* [2] and *quasi-redundancy* [3] concepts emerge from the utilization of components to compensate for a failure, despite not being primarily used with this objective. Replication of control functions over distributed processing units (PUs) is done in such a way, that failed functions are compensated using existing components. These redundancies are implemented reconfiguring the communication routes of the network and PUs.

*Passive* and *active redundancies* replicate the nominal operation of the failed function and *shared-* and *quasi-redundancies* make the failed function operate under degraded conditions. Note that *shared-* and *quasi-redundancies* are a form of *passive redundancies*, i.e., they work only when the primary resource fails.

In order to simplify the nomenclature, we name *homogeneous resources* those needed to perform *passive/active redundancies* where the nominal operation is replicated and *heterogeneous resources* those needed to perform *shared-/quasi-redundancies* where failure of the nominal operation is compensated (i.e., degraded operation).

 ISBN: 978-1-61208-212-7

We reuse existing concepts grouping them to consider the trade-off between the replication of resources and the reuse of existing ones. *Homogeneous resources* lead to increasing the hardware cost. However, the integration and implementation of these resources is not as difficult as with the *heterogeneous resources*. The identification of replaceable functions and the adaptation of the existing architecture to benefit from compatibilities are the main challenges. Therefore, our aim is to complement existing approaches with a method to identify *heterogeneous resources*. This process enables a systematic characterization of the replaceability properties of the system, including those involving sensors and actuators.

### B. Related Works

There is an increasing interest in automotive, avionics and space industry for reusing existing hardware and/or extracting system reconfiguration behaviour.

DySCAS middleware [4] partially addresses our considerations using a context-aware adaptation mechanism, specified by execution and architecture aware contexts. The former context uses distributed policies to detect deviations and react, while the latter embeds meta-information of configuration reasoning (resource dependencies, QoS contracts, compatibility, composability and dependability) within dynamically reconfigurable components. The approach deals with task migrations to cope with hardware failures and network balancing. A global node dynamically maintains for the entire network the intentions of every node and decides the possible configurations based on their requirements. Each node locally performs admission control deciding if a task is schedulable considering resource limitations (memory, CPU, bandwidth) and optimization of resources. The middleware consolidates and disseminates the distributed information.

Adler et al. [5] proposed a component-based modelling and verification method for adaptive embedded systems. The approach aims at exploiting implicitly available redundancies to react to system failures. It provides methodological support for identifying and gathering reasonable system configurations. To do so, each port of the functional component is attached with a quality attribute (QA), which provides means to connect compatible components. Based on QAs, the adaptation behaviour of each component is determined with the required qualities for activation (preconditions) and influences on the provided qualities (postconditions). In order to ensure the causality of the reconfiguration sequences, well-definedness properties are verified by using model-checking and theorem proving techniques.

*Integrated Modular Avionics (IMA)* paradigm defines robust partitioning in onboard avionic systems so that one computing module (Line Replaceable Unit (LRU)) is able to execute one or more applications of different criticality levels independently. The standardized generic hardware modules forming a network leads to looser coupling between hardware and software applications.

SCARLETT project [6] aims at designing reconfigurable IMA architectures in order to mitigate the effect of failures implementing functional and mitigation functions. *Monitoring and fault detection* function aims at detecting component failures. Once a permanent failure is detected, the *reconfiguration supervisor* manages the modifications of configurations given the current configuration and failed module. *Verification activities* check the correctness of the system configuration and the loaded data in the LRU. The centralized supervisor determines a suitable configuration based on a reconfiguration graph, which contains all possible configurations. Reconfiguration policies and real-time and resource constraints, define the set of reachable safe transitions and states. In order to analyse the reconfiguration behaviour when failures occur, a safety model leads to finding the combinations of functional failures. Based on the same concepts, DIANA project [7] aims at distributing these functionalities. This approach improves availability of reconfiguration mechanisms at the expenses of relying on a complex, resource consuming communication protocol.

Based on the potential of the IMA paradigm as a means to provide fault containment strategies, Montano and McDermid [8] presented an autonomous dynamic reconfiguration system. Different information required for an effective dynamic reconfiguration (task scheduling, hardware resources, operating modes, mission objectives, faults and dependability requirements) is gathered based on interactive Constraint Satisfaction Problem theory. The approach divides hard constraints and soft constraints. While the former is compiled off-line the latter can be added and retracted dynamically. Additionally, human interaction is allowed by translating his requirements into soft constraints and weighting the reconfiguration constraints so that higher priority decisions can be controlled.

All these approaches address the integration of reconfigurability and dependability aspects. System reconfiguration requires being aware of the system health, its operating modes as well as the behavioural and dependability requirements. These tasks should be implemented in a (autonomous) centralized or distributed supervisor and coordinated by a communication algorithm. However, the systematic identification of compatible resources has received scant attention. Adler et al. [5] intuitively characterize the quality attributes of the system components so that inter-component compatibilities are identified. To the best of our knowledge, no one is identifying and gathering replaceable resources based on the physical location of hardware components. This viewpoint may provide a useful systematic characterization about the effect of placing components in determined places.

 ISBN: 978-1-61208-212-7

## III. Generic Functional Design Model

The goal of this section is to formalize design concepts so as to provide a systematic consideration of system functions, resources and the relation between them. In order to identify systematically *compatible subfunctions*, this model accounts for the physical location early in the design phase. The system is characterized in a top-down manner, parting from a set of *high-level* (HL) functions (cf. Figure 1).

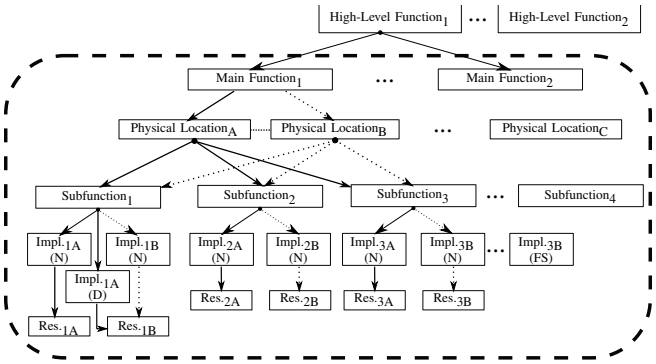


Figure 1. Generic Functional Design Model

Depending on the system HL functions, these are further refined into a set of *main control functions* (MFs). Our design considerations focus on system refinement from MFs downwards to limit the scope of the analysis without losing its generality. The *physical location* (PL) characterizes the place in which these MFs are performed. A single MF may cover different PLs or it may be replicated for each PL (e.g., Temperature Control). A set of *subfunctions* (SFs) define necessary and sufficient means to perform the MFs. Hence, the characterization of the system MFs is specified as follows:

*MainFunction.PhysicalLocation.Subfunction.Implementation*

There may exist different *versions* of SF implementations determined by the availability of resources. The resources provide means to perform a SF using a set of hardware resources, which may allocate software functions. Based on the system means to perform the same SF with different resources, we differ nominal, degraded and fail-safe versions.

The *nominal (N)* version, performs under initial functional design characteristics. The set of Input (I), Control (C) and Output (O) SF components necessary to perform the nominal MF, in conjunction with the necessary resources to address the system requirements, form the nominal design *configuration*.

When the I, C or O subfunction is lost due to the failure of some resource, the configuration to achieve an acceptable outcome may have to change. There may be subfunctions, which provide a *degraded (D)* but acceptable service, even in presence of faults. *Fail-safe (FS)* versions emerge from the need to cope with the insurmountable loss of resources, which result in hazard occurrences. Predetermined solutions should be defined so as to avoid these situations.

According to this classification, we define the concept of *compatible subfunctions*. Two SFs are compatible if their SFs match and they are within a compatible PL. This compatibility would determine the acceptable value for the produced outcome. The compatibility of the PL depends upon the examined SF component. The PL of each SF specify whether we are dealing with a zone-level (e.g., Train.car.zone) or specific-level (e.g., Train.car.zone.location) SF. For I/O SFs performed within the zone-level and depending on the I/O type itself, we consider that the difference (if it exists) between the produced outcomes of compatible subfunctions is acceptable (e.g., Temperature Control in adjacent compartments). However, specific PLs, confine the compatibility within a specific physical space. Generally, output SFs are gathered within this group, due to their specific actuation space. C subfunctions, do not have an explicit dependency on the PL. They are able to perform the C function provided it receives the corresponding I values of the specific PL.

Emerging from these concepts, we make a comparison between *homogeneous* and *heterogeneous resources* (cf. Table I). To do so, we centre on the nominal MF configuration (cfg) and those which use *homogeneous* and *heterogeneous resources*. Remember that since *heterogeneous resources* focus on reusing distributed compatible resources, the MF configuration will vary from the nominal design implementation.

Table I
Comparison between heterogeneous/homogeneous resources and nominal implementation

| Resources | SF | PL | I | C | O | cfg |
|---|---|---|---|---|---|---|
| Homogeneous | = | = | = | = | = | = |
| Heterogenous | = | ≡ | ≡,= | ≡,= | ≡,= | ≡ |

same(=); compatible(≡)

## IV. Reconfiguration Process

The process described throughout this section is based on the application of the generic functional design model (cf. Section III) to model, identify and gather compatibilities and extract customized reconfiguration mechanisms. We rely on a running example in order to discuss and evaluate the process.

### A. Modelling Functions and Resources

The goal of the modelling process is to identify and gather compatible functions performed with alternative resources. The design model, makes these tasks possible, tracing from system functions towards physically distributed implementations. For instance, consider the temperature control ($MF_1$) within a train car (cf. Figure 2).

 ISBN: 978-1-61208-212-7

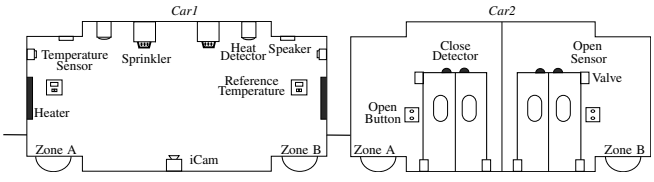


Figure 2. Train Physical Topology

From $MF_1$, temperature measurement, user's temperature reference, temperature control and heating SFs are characterized (cf. Table II).

Table II
CHARACTERIZATION OF TRAIN CAR TEMPERATURE CONTROL

| *Main Function* | *PhysicalLocation* | *Subfunction* | *Implementation* |
|---|---|---|---|
| Temperature Control | Car1.ZoneA | MeasureTemp | $Sensor_A$ |
| | | RefTemp | $RefButton_A$ |
| | | TempControlAlgorithm | PID Control |
| | | Heating | $Heater_A$ |
| | Car1.ZoneB | MeasureTemp | $Sensor_B$ |
| | | RefTemp | $RefButton_B$ |
| | | TempControlAlgorithm | PID Control |
| | | Heating | $Heater_B$ |

If we proceed to model all functions, *heterogeneous resources* are systematically identified. This process consist of matching SF and PL tokens, in order to identify compatible resources.

### B. Identification of Possible Redundancies

The layered modelling of functions, resources and physical locations allows identifying inherent redundancies from multiple system functionalities. The systematic utilization of the described modelling process permits gathering initially different, but suitable, functions.

For instance, consider $MF_1$ for both compartments (zone A, B) of the train car (cf. Table II). Redundancies may appear at input and control SFs. Input redundancies originate from the utilization of a single temperature sensor for different zones. Similarly, contiguous compartment temperature reference values can be used to provide a degraded, but acceptable, subfunction. Control SFs may be implemented on any PU depending on its capabilities (e.g., memory, execution time).

Once *heterogeneous resources* and possible redundancies are identified, a strategy to avoid single point of failures (SPOFs) could be to add *homogeneous resources*. As an example, consider the management of *heterogeneous resources* at the actuator level. Unless actuators are supplied with explicit redundancies or compatible functions actuate in a shared physical space, *heterogeneous resources* become infeasible. Heating SF is not replaceable due to the lack of suitable SFs within the same space. In these cases, *homogeneous resources* should be supplied in order to provide switch over capabilities.

### C. Resource Allocation

When analysing the resource allocation for $MF_1$, some architectural assumptions are made for exemplifying how to customize the approach to a fully distributed system. It is considered that sensors, user temperature references and actuators are distributed in such a way, that they need a PU in order to reach the rest of components using a network communication protocol.

Additionally, even if for the purposes of this example communication, fault detection and reconfiguration failures are not considered, we are expanding the modelling of functions and resources to include them. The idea we are developing is to attach fault detection algorithms to each subfunction so that component and communication failures can be detected. We also need to attach reconfiguration algorithms to each main function, which are responsible for changing the *homogeneus/heterogeneus resources* using the aforementioned fault detection outcomes.

Consequently, one sensor, one reference button, three PUs and a heater connected through a communication network constitute the nominal configuration for the train car temperature control for each zone (cf. Figure 3). Each PU allocates different control algorithms so as to assure functionalities in case of input subfunction failures: Open Loop (OL) algorithm manages the omission of temperature measurement and a default Set Point ($SP_A$) enables handling user's temperature reference failure. Each SF must be allocated to the available resources. To do this, the model defined in Subsection IV-A needs to be completed in order to address the resource allocation decisions:

$$MF_1.\text{Car1.ZoneA.MeasureTemp.Sensor}_A$$
$$MF_1.\text{Car1.ZoneA.MeasureTemp.Sensor}_B$$
$$MF_1.\text{Car1.ZoneA.RefTemp.RefButton}_A$$
$$MF_1.\text{Car1.ZoneA.RefTemp.RefButton}_B$$
$$MF_1.\text{Car1.ZoneA.RefTemp.SP}_A.\text{PU}_{A1,A2,A3}$$
$$MF_1.\text{Car1.ZoneA.TempControlAlgorithm.PID.PU}_{A1,A2}$$
$$MF_1.\text{Car1.ZoneA.TempControlAlgorithm.OL.PU}_{A3}$$
$$MF_1.\text{Car1.ZoneA.Heating.Heater}_A$$

From the previous model, we can observe that there is only one implementation for the Heating SF and therefore, we can deduce that it is a SPOF. This method allows an straightforward identification of SPOFs.

### D. Inference Process and Reconfiguration

Once resource allocation decisions have been adopted, this approach enables the extraction of alternative system configurations. For instance, for $MF_1$ in the train car, the system configurations transit from nominal ($N$) configurations, in which the initial resources are working (W) correctly, to degraded configurations. These configurations allow handling for example the failures of $sensor_A$ ($D_a$),

 ISBN: 978-1-61208-212-7

both sensors ($D_b$), reference button$_A$ ($D_c$) and both reference button failures ($D_d$). Further degradation occurs when the communication network fails ($D_e$). These configurations are illustrated in the Table III. Note that the purpose of this example is not to provide an exhaustive analysis of all existing configurations, but rather we want to illustrate a subset of these configurations, to show the application of the method without losing its generality.

Table III
CONFIGURATION EXAMPLES FOR ZONE A

| *Compatible Tokens: Implementations* | N | $D_a$ | $D_b$ | $D_c$ | $D_d$ | $D_e$ |
|---|---|---|---|---|---|---|
| $MF_1$.Car1.ZoneA.MeasureTemp.Sensor$_A$ | W | F | F | W | W | F |
| $MF_1$.Car1.ZoneA.MeasureTemp.Sensor$_B$ | | W | F | | | F |
| $MF_1$.Car1.ZoneA.RefTemp.RefButton$_A$ | W | W | W | F | F | F |
| $MF_1$.Car1.ZoneA.RefTemp.RefButton$_B$ | | | | W | F | F |
| $MF_1$.Car1.ZoneA.RefTemp.SP$_A$.PU$_{A2}$ | | | | | W | F |
| $MF_1$.Car1.ZoneA.RefTemp.SP$_A$.PU$_{A3}$ | | | | | | W |
| $MF_1$.Car1.ZoneA.TempControlAlg.PID.PU$_{A1}$ | W | W | | W | W | F |
| $MF_1$.Car1.ZoneA.TempControlAlg.OL.PU$_{A3}$ | | | W | | | W |
| $MF_1$.Car1.ZoneA.Heating.Heater$_A$ | W | W | W | W | W | W |

We need to assign priorities to each implementation for the SF components. This allows an automated generation of configurations in the case of failure occurrences, e.g., the priorities for the MeasureTemp SF for $MF_1$.Car1.ZoneA would be $\{\text{Sensor}_A, \text{Sensor}_B\}$.

Figure 3 shows the network and the data flow between different PUs. The thick lines represent the physical measurements of the sensors. The solid lines represent the data transfers in nominal operation mode. The dashed lines represent data transfers in a degraded operation mode when $PU_{A1}$ and $PU_{B1}$ fail. Finally, the dotted lines with empty arrowheads represent the data transfers in a degraded operation mode when $PU_{A1}$, $PU_{B1}$, $PU_{A2}$ and $PU_{B2}$ fail. In this configuration, $PU_{A3}$ and $PU_{B3}$ manage the temperature using OL algorithms and default reference temperature values.

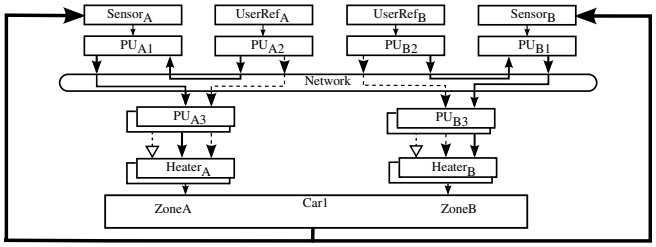


Figure 3. Train Car Logical Reconfiguration Topology

## V. CONCLUSION AND FUTURE WORK

In this paper, a dependable design strategy has been sketched. Dedicated replication of system components satisfy the avoidance of SPOFs. However, in some environments it is feasible and desirable to make use of existing resources so that other compatible functions are supplied with *heterogeneous resources* (e.g., trains, buildings).

Identification and implementation of the proposed reconfiguration strategies will incur an extra cost. However, for a given dependability level, this approach would reduce the overall system hardware cost as *heterogeneous resources* are systematically identified. Note that this is only true in the previously mentioned environments.

We are developing the approach to support fault detection and reconfiguration. This will complete the method and we will be able to evaluate the dependability gains and cost trade-offs of this approach. Kazman et al. [9] propose a trade-off analysis method, which enable the architectural dependability-cost evaluation. This method could be integrated with this approach. The final goal is to obtain a complete method, where a quasi-optimal solution is obtained in a massively networked scenario.

In the cases where the dependability is critical, greater architectural details should be considered (e.g., power supplies, communication routes). This would allow evaluating the needed *homogeneus/heterogeneous resources*.

## REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, pp. 11–33, January 2004.

[2] J. Wysocki, R. Debouk, and K. Nouri, "Shared redundancy as a means of producing reliable mission critical systems," in *Proc. of RAMS'04*, 2004, pp. 376 – 381.

[3] J. Galdun, J. Ligus, J.-M. Thiriet, and J. Sarnovsky, "Reliability increasing through networked cascade control structure - consideration of quasi-redundant subsystems," in *IFAC Proc. Volumes*, vol. 17, 2008, pp. 6839–6844.

[4] R. Anthony, D. Chen, M. Törngren, D. Scholle, M. Sanfridson, A. Rettberg, T. Naseer, M. Persson, and L. Feng, *Autonomic Communication*. Springer, 2009, ch. Autonomic Middleware for Automotive Embedded Systems, pp. 169–210.

[5] R. Adler, I. Schaefer, M. Trapp, and A. Poetzsch-Heffter, "Component-based modeling and verification of dynamic adaptation in safety-critical embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 20:1–20:39, Dec. 2010.

[6] P. Bieber, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, "Preliminary design of future reconfigurable ima platforms," *SIGBED Rev.*, vol. 6, no. 3, 2009.

[7] C. Engel, A. Roth, P. H. Schmitt, R. Coutinho, and T. Schoofs, "Enhanced dispatchability of aircrafts using multi-static configurations," in *Proc. of ERTS'10*, 2010.

[8] G. Montano and J. McDermid, "Autonomous and/or interactive constraints-based software reconfiguration for planetary rovers," in *Proc. of ASTRA'08*. ESA/ESTEC, 2008.

[9] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for Architecture Evaluation," SEI, Carnegie Mellon University, Tech. Report CMU/SEI-2000-TR-004, 2000.

 ISBN: 978-1-61208-212-7 

# Security Control Variations Between In-house and Cloud-based Virtualized Infrastructures

Ramaswamy Chandramouli
*Computer Security Division, Information Technology Laboratory*
*National Institute of Standards & Technology*
*Gaithersburg, MD, USA*
*mouli@nist.gov*

***Abstract*-Virtualization-related components (such as Hypervisor, Virtual Network and Virtual Machines (VMs)) in a virtualized data center infrastructure need effective security controls. However, the differences in scope of control (among stakeholders) over this component set between in-house and cloud-based virtualized infrastructures introduce variations in security control measures that can be deployed by the respective stakeholders. In this paper, we analyze those variations and their efficiency and security impacts. We also suggest technology enablers that can minimize those impacts and improve the overall security robustness of the basic computing units of a virtualized infrastructure, (i.e.,VMs).**

***Keywords-Virtual Machine; Virtual Network; Hypervisor; Virtualized Host; Cloud Service Model***

## I. INTRODUCTION

Server Virtualization, in some instances augmented with storage virtualization, is becoming the trend for data center infrastructures in both enterprises and cloud provider environments. In fact, Virtualized Servers and Virtualized storage have become the platform for many enterprise applications such as Enterprise Resource Planning (ERP) [1] because of the following:

(a) Efficiency in the utilization of processor and memory resources in a virtualized host because of the ability to run multiple Virtual Machines (VMs) as opposed to a non-virtualized host where only a single O/S stack can be run. Similar efficiencies can be achieved in the case of virtualized storage because of the presence of an abstraction layer above the physical storage layer, (e.g., disk arrays).

(b) Scalability and Elasticity that are enabled in Virtual Servers and Virtual Storage by the very nature of virtualization. An example is the capability to add VMs at will to a physical host with underutilized capacity and ability to add disk arrays transparent to the programs that gather, store, retrieve and process data.

There is general agreement in the security community that the security control measures used for protecting servers that run a single O/S stack (referred to as non-virtualized hosts) alone are not sufficient for protecting servers that run multiple O/S stacks (referred to as virtualized hosts). The reason for the agreement is the presence of a single trusted layer, (i.e., the hypervisor) below multiple VMs in virtualized hosts and the risk of compromise to this layer posing the risk of compromising the integrity of all VMs running in that host [2]. The detailed differences between virtualized hosts and non-virtualized hosts are given below:

(a) In a non-virtualized host, the interface to the hardware is through a regular O/S, whereas in a virtualized host, the interface to the hardware is through a software module, called a hypervisor, which contains just the kernel of an O/S with some necessary additions such as device drivers, etc.

(b) A virtualized host has resident in it multiple Virtual Machines (VMs), each with its own stack of O/S and Applications. All of the VMs share the same physical resources provided by the virtualized host – such as the processor, memory and directly attached storage. The hypervisor mediates access to shared resources by the various VMs, and provides isolation between the VMs.

(c) To enable VMs to communicate to the physical network and to provide isolation among them, a Virtual network is defined within each virtualized host. A Virtual network can be looked upon as a set of logical (sub)networks within a shared physical network. A virtual network can be configured using a combination of software-defined communication interfaces called virtual network interfaces (or vNICs) inside a VM as well as software-based switches, called Virtual Switches,that can be defined within the hypervisor.

In a virtualized enterprise data center, catering to internal information technology (IT) processing needs of an

 ISBN: 978-1-61208-212-7 

enterprise (henceforth referred to as in-house virtualized infrastructure), all the components of a virtualized host are owned and controlled by the single entity, i.e., the enterprise. However, in the virtualized data center owned and operated by cloud service providers (henceforth referred to as cloud-based virtual infrastructure), while the virtualized host (the physical machine) and the software that provides the virtualization, (i.e., the hypervisor) are owned by the cloud service provider, the VMs in it are created and operated by the cloud service consumer. Hence, the internal configuration of VMs in a cloud-based virtual infrastructure belonging to an Infrastructure as a Service (IaaS) cloud provider is under the control of the cloud service consumer although the capabilities to configure a virtual network linking these VMs will still rest with the cloud service provider. Thus, we see that there are differences in the scope of control over the components of a virtualized infrastructure between in-house and cloud-based virtualized environments.

The main objective of this paper is to illustrate the variations in security control measures between the in-house and cloud-based virtualized infrastructures that these scope of control differences introduce. A second objective, or rather a by-product of the illustration process is to show the impact of these variations (in security control measures) on the effectiveness and efficiency of the total set of security controls and how they can be addressed to improve the overall security robustness of the basic computing units of a virtualized infrastructure, (i.e., the VMs).

The organization of this paper is as follows. In Section II, we identify the differences in scope of control among stakeholders over the components of a virtualized infrastructure by looking at the layers of a cloud service architectural stack. Section III identifies threat scenarios relating to virtualization-related components. In that section we also look at the broad class of security control measures and identify whether these control measures may be affected by differences in scope of control between in-house and cloud-based virtualized infrastructures. Section IV describes in detail the security control variations for VM protection at the Virtual Network layer and VM end-point. Section V provides the summary and conclusions.

## II. SCOPE OF CONTROL IN CLOUD-BASED INFRASTRUCTURES

In the case of in-house virtualized infrastructures, since all components are owned and operated by a single entity, i.e., the enterprise, differences in scope of control over the overall set of components do not arise. Hence we limit the scope of control analysis only to cloud-based virtualized infrastructures. In order to do that we digress a bit and look at the broad picture of cloud service models. The three widely accepted cloud service models are [3]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), all of them consisting of two major players – the cloud provider and the cloud consumer. In the case of SaaS, the cloud provider makes available a software application while PaaS offers a set of development tools (or an application development environment such as J2EE [4] or .NET [4]) to develop and possibly host applications. In these two service offerings, (i.e., SaaS and PaaS) the underlying data center infrastructure used (or leased from another provider) by the corresponding category of cloud service providers need not be based on virtualized servers. In the case of IaaS, what is made available to the cloud consumer are computing units in the form of VMs. Hence, the data center infrastructure in the case of IaaS cloud service providers should consist of only virtualized hosts. However, in this paper, we assume that the data center infrastructure of a cloud service provider irrespective of the cloud service model (because of efficiency, scalability and elasticity considerations) is a virtualized one consisting of virtualized hosts, virtual network and VMs. Based on this assumption, we can start taking a look at the various layers in a cloud service architectural stack. One such architectural stack based on slight variations from the model given by the Cloud Security Alliance [5] is given in Figure 1 below. In this stack, we notice that the facility, networking infrastructure and the physical host layers are common to all IT infrastructures - whether virtualized or not- and hence these layers are not relevant for our scope of control analysis. Going up one more layer in the stack, we find that it is in the resource abstraction layer that the main engine providing the virtualization, (i.e., the hypervisor) and virtual network are defined. Virtual Machines - the main computing units of a virtualized infrastructure reside in the VM layer. Our focus of attention for identifying the differences in scope of control between in-house and cloud-based environments is limited to these two layers. This is due to the fact that these are the two layers whose composition differs between virtualized and non-virtualized infrastructures. We give below our observations on the scope of control among stakeholders over components in cloud-based virtualized infrastructures used in all three cloud service-models.

 ISBN: 978-1-61208-212-7 

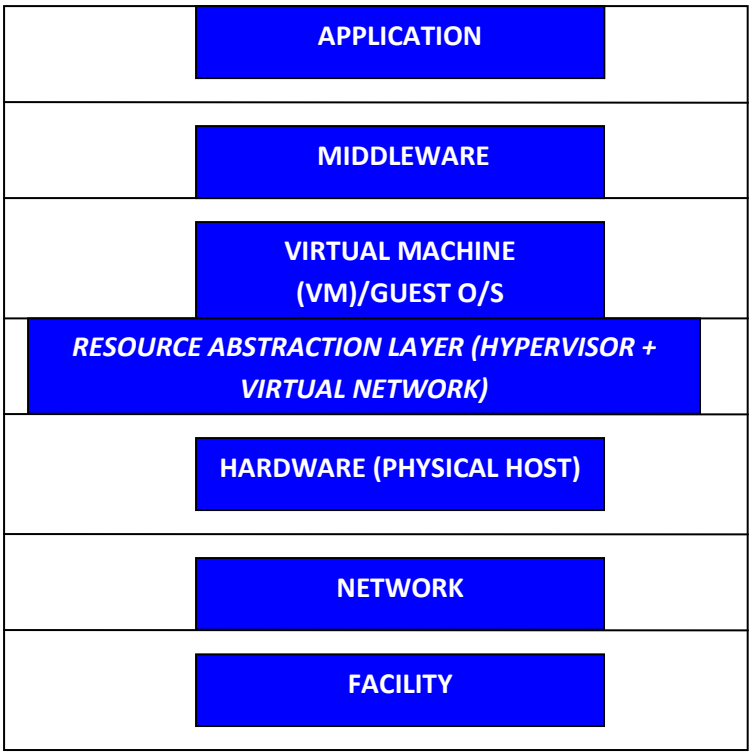


Figure 1. Cloud Service Layers.

(a) In the resource abstraction layer (virtualization layer), all components - the hypervisor and the virtual network (in all three cloud service models) - are totally under the control of only one entity, (i.e., the cloud provider).

(b) In the case of VM layer, the single component it holds - the VM instance with its embedded Guest Operating System -is under the control of cloud service provider in the case of SaaS and PaaS service models but controlled by the cloud service consumer in the IaaS model (mainly due to the fact that the SaaS provider does not want to assume any administrative responsibility for it after a VM instance is configured and instantiated by an IaaS consumer).

Although not shown in the architecture diagram, conceptually, one can think of a data layer which contains the components for the management software and the physical artifacts for storing the data that applications generate and use. This layer is entirely under the control of a single entity, i.e., the cloud provider - in all three cloud service models and hence is not relevant for our scope of control analysis. It is worth mentioning that although all storage-related technologies (both virtual and physical) are under the control of cloud service providers, the responsibility for appropriate security control measures for data protection (through encryption of data in transit and data at rest - for the portion of data generated and used) still rests with the cloud consumer [6].

## III. THREAT SCENARIOS & SECURITY CONTROL MEASURES FOR ENTIRE VIRTUALIZED INFRASTRUCTURES

Our scope of control analysis narrowed our focus to just two layers that contain all the artifacts relating to virtualization, i.e., the Resource Abstraction layer and the VM layer. Hence, our threat analysis is also limited to scenarios involving the components contained in these layers. These components are re-listed here for facilitating further discussion:

(a) Hypervisor and Virtual Network - from the Resource Abstraction Layer

(b) Virtual Machines (VMs) with their Guest O/S - from the VM Layer

 ISBN: 978-1-61208-212-7 

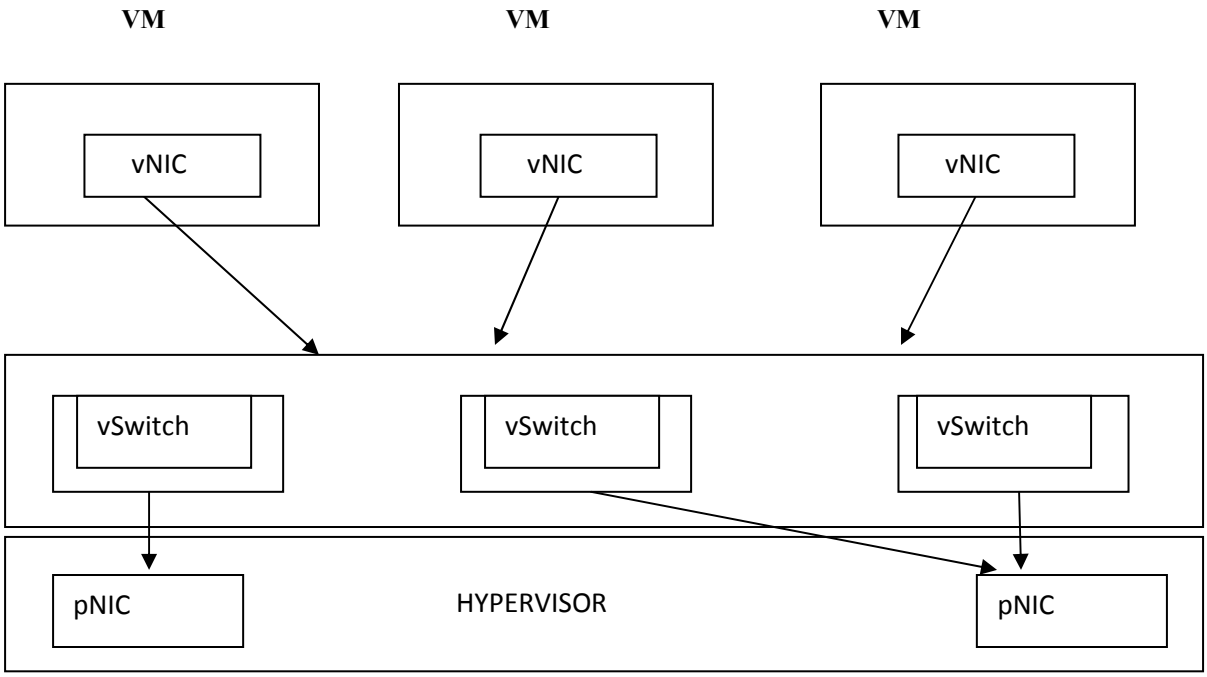


Figure 2. A Virtual Network Configuration.

The virtualization environment involving the above components can be described as follows. The VMs provide the complete, encapsulated computing stacks built up of Guest O/S with middleware and applications riding on top of the chosen O/S. A network linking these VMs , thus enabling communication among them is called a Virtual Network. An example of a virtual network configured using software-defined virtual network interface card (vNIC) within each VM and software-defined virtual switches (vSwitch) defined within the hypervisor is shown in Figure 2. A virtual network provides communication not only among VMs residing on a single virtualized host but also connectivity with the outside world (the physical network), if any of the virtual switches is also connected to a physical network interface card (pNIC) of the virtualized host. In this virtualization environment, the most common (though not exhaustive) threat scenarios are identified below:

TS-1: The compromise of the hypervisor (by exploiting the vulnerabilities in the kernel - which is rare) can potentially compromise the security of multiple VMs (in fact potentially all) resident on that virtualized host [7].

TS-2: A single VM that has been compromised can be used as a launching pad for attacking other VMs (especially if they share some common resources such as memory or there exists a communication channel between them due to the fact that these VMs constitute the different tiers (webserver, application server , database server etc) of a multi-tier application) [8].

TS-3: Normally, the effect of all operations performed within a VM must be restricted to that VM. However, under some circumstances, a malicious program running in a VM, by exploiting vulnerabilities in the hypervisor software, can alter the state of other VMs, the hypervisor itself or even the hardware. Such a VM is called a rogue VM and it poses the threat of subverting the isolation property required to be provided by the hypervisor.

TS-4: Attacks on running guest VMs by a malicious hypervisor. The impact of this threat is the same as that of TS-3 but the threat source here is the hypervisor as opposed to a rogue VM in the case of TS-3.

For each of the threat scenarios, let us see the broad class of security control measures that are required and identify whether these control measures may be affected by differences in scope of control among stakeholders between in-house and cloud-based virtualized infrastructures.

TS-1 has to do with the compromise of the hypervisor. The usual security control measure adopted is to keep the

 ISBN: 978-1-61208-212-7 

patches from the hypervisor vendor up to date. This is a hypervisor-based security control measure and since the hypervisor is always under the control of the single entity in both in-house and cloud-based virtualization infrastructure, the potential for variation in security control measures due to differences of who has control does not arise.

TS-2, TS-3 & TS-4 have to do with compromise of the VM. Security control measures can be provided for VMs through a combination of: (a) Virtual Network-resident security controls and (b) VM-resident security controls [9]. In the case of in-house and SaaS/PaaS cloud-based virtualized infrastructures, both the Virtual Network and the VMs are under the control of a single entity. However, in the case of IaaS cloud-based virtualized infrastructure, the VMs to be protected are under the control of IaaS cloud consumer. This is where there is potential for variations in control measures for protection of VMs between in-house and cloud-based virtualized infrastructures. These variations are analyzed and discussed in the next section.

## IV. SECURITY CONTROL VARIATIONS FOR VM PROTECTION

Security controls for VM protection can be deployed both at the Virtual Network layer as well as in the VMs themselves. As we have already seen, both these layers are accessible to a single entity only in the case of in-house and SaaS/PaaS cloud-based virtualized infrastructures. However, only the Virtual Network layer is accessible for IaaS cloud provider and the VMs are accessible for IaaS cloud consumer. Hence variations in security controls for protection of VMs are introduced due to these differences in scope of control among stakeholders as discussed below:

### *A. Security Control Variations at the Virtual Network Layer*

In virtualized infrastructures for in-house IT as well as for providing cloud services (all three service models), the configuration of a Virtual Network (the network linking all virtual machines within a single virtualized host) is entirely under the control of the data center owner/operator. Leveraging the virtual network, there are two approaches to providing security for VMs. They are [10]:

(a) Extending the concept of Virtual LAN (VLANs) into the virtual network and

(b) Virtual Network Configuration-based solutions for VM protection

In the VLAN-based approach, the virtual switches defined in a hypervisor are made to recognize the VLAN tags and thus the concept of network isolation in the physical network is extended to the virtual network inside a virtualized host. As far as Virtual Network Configuration-based solutions go, there are two types: In the first approach, VMs hosting sensitive resources such as fileshare VM or hosting sensitive applications such as data warehousing or payroll processing are connected to isolated virtual network segments which are not behind any firewall or Network Address Translation devices. In the second approach, a special-purpose VM called a Virtual Security Appliance [11] that contains a hardened Guest O/S and one or more security applications, is installed and configured to provide the necessary protection for VMs. The type of protection depends upon the security application(s) that is (are) packaged as part of the Virtual Security Appliance. Examples of popular security applications are Firewall and Intrusion Prevention [12]. In a firewall solution, the data center operator can provide protection to VMs by defining VM-specific rules that can control traffic to and from virtual machines. There are tools in the market place [11], which, using a combination of a management server and a set of security appliances, can control traffic in and out of an arbitrary set of VMs irrespective of the VLANs to which they belong. By placing the entire set of virtual machines to be protected on an internal-only virtual switch of the virtual network, all traffic is made to flow through the security appliance and thus the security appliance can act as a Layer 2 bridge that controls all traffic flowing to and from the protected VMs without reconfiguring them in any way. The consequence of this is that firewall rules encompassing layers 2, 3 & 4, (i.e., including IP addresses and specific TCP or UDP port) can all be defined using this virtual network-based security appliance. However, in the case of an IaaS Cloud consumer, who owns and operates a set of VMs, the virtual network layer is not under his/her control. Hence a virtual network-based firewall cannot be deployed in this situation and this class of user can only provide the necessary traffic control by having a VM-based firewall. Deploying a VM-based firewall solution imposes a great deal of performance overhead compared to a virtual network-based firewall as this security application competes for the same resources, (i.e., CPU, Memory, etc.) as functional (business) applications do on each of the VMs. For example, if there are 10 VMs in a virtualized host, 10 firewall applications will be competing for its

 ISBN: 978-1-61208-212-7 

resources. On the other hand, providing the same firewall functionality using a Virtual Security Appliance will involve running a single instance of a firewall application instead of ten (in the case of VM-based solution), thus providing a significant performance improvement while accomplishing the same security objective of blocking unwanted/malicious traffic in and out of the VMs that the cloud service consumer has rented and is operating.

### *B. Security Control Variations at the VM end-point*

The VMs (often called the endpoints) in a virtualized infrastructure need to have the same protection measures as a physical server in a non-virtualized environment. These protection measures include but not limited to [8]:

- Anti Virus/Anti Malware Software
- Intrusion Prevention

In the case of a virtualized infrastructure providing IaaS cloud service, the VMs are owned and operated by cloud consumers and hence the security of these VMs rest with them. Because of the fact that the hypervisor controlling these VMs is owned and under the control of the IaaS cloud provider, the only security control measure available to the cloud consumer is to run individual instances of above classes of software (anti-virus, etc.) in each of the VMs rented and operated by them.

In the case of virtualized infrastructures providing in-house IT needs or providing SaaS or PaaS cloud services, both the hypervisor and the VMs are owned and operated by a single entity - data center owner or operator. Using the published interfaces (called the hypervisor introspection APIs) provided by the hypervisor vendor, the data center owner/operator can either develop (or procure from a third-party), a security application that can be installed as a security appliance on a special hardened VM. This security appliance thus runs as a single instance of security software and this instance would be running separate from all other instances of Guest O/Ss running in the various VMs that it would protect. For example, a security appliance for an antivirus solution can perform the functions of - memory scanning, monitoring of processes and investigation of network traffic - for all VMs and Guest O/Ss running on that virtualized host. The variations in security control measures that can be deployed by the stakeholder between the two virtualized infrastructure environments, (i.e., IaaS cloud Versus SaaS/PaaS cloud) for VM protection has the following efficiency and security implications:

(a) Multiple instances of say (an anti-virus solution) in a single physical (virtualized) host makes of the order of magnitude huge demands on the processor and memory cycles of that host as opposed to a single instance of security software. Related to this is the management issue of keeping these security solutions in synch in all of the VMs in a virtualized host.

(b) A rogue process within a Guest O/S can potentially shut down the anti-virus solution running in that same Guest O/S. However, the single instance of such a solution running in a security appliance in a hardened VM that runs in the same virtualized host is not only able to thwart such attacks on itself, but is also able to provide protection to all other VMs running in that virtualized host.

## V. CONCLUSION

In this paper, we saw that protection of VMs can be obtained through efficient and effective means in the case of in-house and SaaS/PaaS cloud-based virtualized infrastructures through the following:

- Implementing VLANs in the virtual network
- Creating isolated network segments for VMs running sensitive applications, and
- Virtual Security Appliances utilizing hypervisor introspection API

However, the protections provided by the above measures have to be obtained through a less efficient way by means of individual VM-based solutions in the case of IaaS cloud-based virtualized infrastructures. This is due to the fact that the IaaS cloud consumer who needs to protect VMs does not have access to components of Resource Abstraction layer such as the Hypervisor and the Virtual Network. Analysis of such variations in security control measures and their relative effectiveness and efficiency impacts can lead to exploration of ways to minimize those impacts.

Specifically, to address the effectiveness and efficiency gap between security controls available for stakeholders in the in-house and cloud-based infrastructures, we propose the following:

(a) Provide selective visibility to cloud customers to VMs rented by them through hypervisor introspection API

(b) Define pre-defined VLAN segments for cloud customers to place the VMs created/rented by them..

The justification for the above measures to improve the overall security robustness of VMs can be made based on the following observations:

 ISBN: 978-1-61208-212-7 

(a) Any security/monitoring solution based on virtual network can significantly reduce the demand on the CPU cycles of the virtualized host compared to a host-based (i.e., VM-based) solution

(b) The vulnerability of the Guest O/S itself is not a factor with respect to the integrity of the security solution as these types of solutions that are based on visibility into the virtual network are run on dedicated VMs with hardened Guest O/Ss.

## REFERENCES


[1] M.Hoyer, K.Schröder, Daniel Schlitt, and D. Wolfgang Nebel, "Proactive Dynamic Resource Management in Virtualized Data Centers", ACM Conference on e-Energy, New York, NY, USA, May 2011.

[2] T. Garfinkel and M. Rosenblum, "When Virtual is harder than Real: Security Challenges in Virtual Machine Based Computing Environments" ,Stanford University Department of Computer Science. http://www.stanford.edu/~talg/papers/ HOTOS05/virtual-harder-hotos05.pdf [Retrieved: July, 2012]

[3] P. Mell and T. Grance,"*A NIST Definition of Cloud Computing," NIST SP 800-145*, http://csrc.nist.gov/publications/ #SP- 800-145, [Retrieved: May, 2012]

[4] D.M. Whittinghill, and K.D. Lutes, "Teaching Enterprise Application Development: strategies and challenges", ACM SIGITE' 11 Conference, West Point, NY, Oct 2011.

[5] Cloud Security Alliance, "*Security Guidance for Critical Areas of Focus in Cloud Computing, v2.1*," www.cloudsecurityalliance.org/ csaguide.pdf, pp. 61-64.

[6] L.M. Kaufman,"Data Security in the world of cloud computing." IEEE Security and Privacy, Vol. 7, No. 4, 2009

[7] S. Jin, J.Ahn, S.Cha, and J.Huh, "Architectural Support for Secure Virtualization under a Vulnerable Hypervisor, "ACM MICRO'11 Conference, Porto Alegre, Brazil, Dec 2011, pp. 272-283.

[8] J. Sahoo, S.Mohapatra, and R.Lath, "Virtualization: A Survey On Concepts, Taxonomy And Associated Security Issues," IEEE 2nd International Conference on Computer and Network Technology, Bangkok, Thailand, Apr 2010, pp. 222-226.

[9] J. N. Matthews, 'et al.''Running Xen – A Hands-On Guide to the Art of Virtualization," Prentice Hall, 2008

[10] Five exciting VMware networking features in vSphere 5 http://searchvmware.techtarget.com/tip/Five-exciting-VMware-networking-features-in-vSphere-5 [Retrieved: March, 2012]

[11] S. Lowe "Mastering VMware vSphere 4," Wiley Publishing, 2009.

[12] L. Garber, "The Challenges of Securing the Virtualized Environment", IEEE Computer, Volume 45 Issue 1, Jan 2012.


 ISBN: 978-1-61208-212-7 

# Towards a State-driven Workload Generation Framework for Dependability Assessment

Domenico Cotroneo, Francesco Fucci, Roberto Natella
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli Federico II
Via Claudio 21, 80125, Naples, Italy
Email: {cotroneo,francesco.fucci,roberto.natella}@unina.it

***Abstract*—Assessing dependability of complex software systems through fault injection is an elaborate process, since their fault tolerance is influenced by the *state* in which they operate. This paper focuses on the definition of *state-driven workloads* in fault injection experiments, that is, workloads that bring a system in a given target state to be evaluated. We discuss a framework for the automated generation of state-driven workloads, and provide a preliminary evaluation in the context of the Linux 2.6 OS.**

***Keywords*-Fault Injection, Fault Tolerance, Stateful Systems**

## I. Introduction

Fault injection, that is, the deliberate introduction of faults into a system, is an approach for providing evidences that a system is tolerant to faults in the environment and in the system itself. The increasing complexity of software systems makes fault injection an elaborate process, since these systems can operate in several different conditions, namely *states*, which have influence on their dependability. The importance of the state for dependability assessment purposes is emphasized by several studies on dependability assessment of stateful complex systems, such distributed filesystems [1], DBMSs [2], and multicast and group membership protocols [3], [4], [5]. In fact, when complex software systems are evaluated both *the activation and manifestation of faults*, as well as *the ability of the system to tolerate them*, depend on the system state, such as the current step of a numerical algorithm or of a distributed protocol [6], [7], [8]. For instance, this is the case of Mandelbugs [9], that is, a class of software faults that manifest themselves depending on the system state and on complex interactions with the hardware, the OS and other software in the system. Therefore, fault injection experiments have to be carefully planned by including the system state [10].

To take into account the state, a fault injection experiment should adopt an appropriate *workload*, which is the set of requests that are being submitted to the system when a fault is injected. A *state-driven workload*, i.e., a workload that brings the system in a given state during the experiment, is important to assure the significance and the efficiency of experiments, by *avoiding faults that do not actually manifest themselves* and by *covering every state in which fault tolerance mechanisms need to be tested*.

The generation of workloads for complex software systems is an open issue. Past studies proposed the generation of synthetic randomly-generated workloads (e.g., random CPU and I/O operations) according to a given distribution [11], [12], but this approach does not allow to bring a system into certain states that can be hard-to-reach, which therefore cannot be exercised and analyzed. Other fault injection approaches rely on hand-written workloads developed by testers [13], [14], [10], [15]. However, complex software systems are difficult to control since the relationship between the workload and states is complex and non-deterministic, due to the effect of random factors such as process scheduling and I/O delays. Therefore, it is still a difficult and time-consuming task to define a workload that brings the system in a desired state.

This paper is a first step towards the automated generation of state-driven workloads in complex software systems, which is an open research issue. We discuss a general framework for the purpose, and present a preliminary experiment on a complex system. The proposed framework is based on a closed-loop paradigm: a *workload generator* explores the space of possible workloads, and a positive feedback is provided to the workload generator if the system is approaching the target state, until the system converges to this state. The approach is fully automated, as it is only based on the feedback it receives from the controlled system, and does not rely on *a priori* knowledge about the relationship between workloads and states. The feasibility of the approach and its ability to reach a target state were evaluated in a preliminary case study, in which the workload generator is adopted to control the state of the Linux 2.6 SMP scheduler.

The paper is organized as follows. Section II discusses relevant fault injection techniques and tools for stateful software systems. Section III describes the proposed state-driven workload generation framework, which is further detailed in Section IV. Section V discusses a case study and some preliminary results. Section VI concludes the paper.

## II. Related work

Dependability attributes of stateful systems have been studied in fault injection studies since a long time. For this reason, theoretical frameworks have been proposed in past works to make the fault injection process systematic. In [16], [17], [18], formal testing approaches for fault-tolerant systems are proposed, based on *state models* that describe the expected behavior with respect to normal inputs and faults (e.g., communication or memory faults). The model is used to generate

 ISBN: 978-1-61208-212-7

and monitor events for testing the actual implementation of the system, and to assess the coverage of the tests. The role of the system state was also emphasized by [19], in which faulty inputs are injected at the interface between drivers and the OS. It is showed that if faults are injected at different times during a sequence of interface function calls, then a higher number of vulnerabilities is discovered than injecting at the first occurrence of the target function call.

Tools have also been developed to aid testers in fault injection experiments in stateful systems. FTAPE [11], [12], a tool for injecting CPU, memory, and I/O faults in fault-tolerant computers, provides support for generating synthetic stressful workloads, based on the observation that a stressful workload is able to increase the percentage of injected faults that are activated and that propagate through the system, and thus are useful to assess the system response to faults. Faults are injected when the stress level of CPU, memory, or I/O (i.e., CPU utilization and memory and I/O throughput) are higher or lower than a threshold. To control the system state, FTAPE spawns a set of one or more processes that are CPU-intensive (by performing arithmetic operations), memory-intensive (by performing sequential reads and writes to large memory areas), and I/O-intensive (by repeatedly performing file opens, reads, writes, and closes), and lets testers to specify the distribution of CPU, memory, and I/O operations.

The ORCHESTRA tool [13], [20], aimed at testing distributed real-time systems, adopts a *script-driven probing* approach, in which messages exchanged by a process are intercepted and analyzed by a *protocol fault injection* layer (PFI) in the OS kernel, which identifies the current state of the protocol under test and corrupts/delays messages to perform fault injection. In order to track the protocol and to trigger fault injection, the PFI layer executes a *script program* provided by the tester, which describes the protocol under test using a state machine specification. In a similar way, the FCI fault injection framework for grid computing systems [15] provides a language specifically developed for specifying *fault injection scenarios*, which is used by the tester to describe *commands* to be sent to processes in the system (e.g., for stopping, halting, or resuming process execution) and *guard conditions* that trigger commands. NFTAPE [14] is a *portable* fault injection tool for distributed systems (i.e., it allows to easily customize fault injection for different systems and fault models), which introduced the concept of *LightWeight Fault Injector*, i.e., a small program running in the target system that is invoked by a remote controller to inject a fault, and that embeds the logic needed to implement a fault model for a given target platform.

The Loki tool [10] addressed the important problem of performing fault injection based on the *global state*, i.e., the condition that triggers fault injection is based on several nodes of the distributed system. Due to the problem of *clock* synchronization at each node and to message delays, a fault may be injected in a global state that is different than the desired target state. To mitigate this problem, Loki performs an off-line analysis of execution traces in order to discard experiments in which fault injection is likely to have been triggered in a wrong state.

A limitation of the tools mentioned above is that the workload has to be manually tuned in order to bring the system in a desired state. In the case of FTAPE, the tester selects the distribution of CPU, memory, and I/O operations generated by the synthetic workload. In the case of other tools, such as ORCHESTRA, FCI, NFTAPE, and Loki, the tester specifies a fault injection scenario by means of state machines, which are used by the fault injection tool to track the current state and trigger a fault when a desired state is reached. These approaches assume that the system is excited by a workload able to bring the system in the target state. However, devising such a workload is a tricky task for complex systems, since the tester has to carefully define the timing and the order of messages or inputs to be sent to the system. This problem is further complicated by the inherent non-determinism of complex systems, which makes difficult to bring the system in the target state and does not assure a correct execution of the fault injection experiment. This paper represents a further step towards solving this research problem.

## III. Overview

Our framework for workload generation is composed by three subsystems, namely: (i) the System Under Test (SUT), which is the target of the experimental dependability evaluation, (ii) the Workload Generator (WG), which submits inputs to the SUT in order to provide a workload and monitors its behavior, and (iii) a Fault Injector (FI), which is adopted to inject faults into the SUT. The problem considered in this paper, namely *state-driven workload generation*, can be formulated as follows: *given an initial state $s_0$ of the SUT, and a goal state $s_g$ in which an experiment has to be performed, the WG has to find a sequence of actions that drives the SUT from $s_0$ to $s_g$*. To this purpose, the WG and the SUT are put in a closed-loop configuration (shown in Fig. 1), in which the WG sends inputs to the SUT and collects information about its state. When the SUT is in the target state $s_g$, the WG triggers the FI, and then the fault injection experiment is performed.

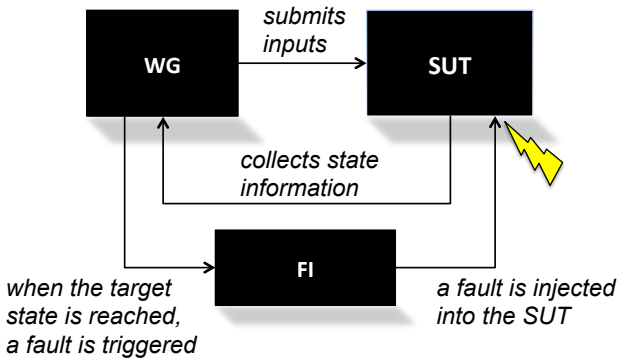


Fig. 1: Overview of the workload generation framework.

The WG evaluates how the SUT is behaving by computing a feedback function $D$ based on state information. An increase

of the feedback value means that the action $a_t$ has driven the SUT to a state "close" to the goal state; otherwise, the function decreases. Actions to be performed by the WG may consist in messages or commands sent to the SUT, or in variations in the rate or type of inputs generated by the WG. The WG aims to find a sequence of actions $s =< a_1 \dots a_t >\in S$ that brings the SUT *as close as possible to the goal state*, i.e., after performing the actions in $s$, $D$ reaches a peak value at time $t$:

$$s = \arg\max_{\hat{s} \in S} D(t) \ . \quad (1)$$

An example of stateful system in which such a framework could be deployed is represented by a *File System*, since several past studies on fault injection and robustness testing have investigated dependability of File Systems with respect to faulty disks or applications [1], [21], [22]. In this kind of system, the state could include aspects related to I/O transactions, such as the amount of cached data that still has to be flushed to disk and the available disk bandwidth. A dependability analysis of a File System could aim to assess the probability of persistent data corruption due to faults, which in turn depends on the state of the system (e.g., the amount of cached data). In order to bring the File System in a given target state, the Workload Generator should perform I/O operations or instantiate new I/O-bound processes until the target is reached. Tuning a state-driven workload in such a scenario can be a tricky task due to complex interactions between the File System, other OS subsystems, user applications, and I/O devices, and automated workload generation is useful to relieve the tester of this task.

## IV. The proposed framework

In this section, we will give a description of the framework focusing on its key elements: (i) state modeling, (ii) actions, (iii) reward function, (iv) search algorithm. The framework is meant to be tailored to the specific SUT, by using an appropriate definition of these elements. We discuss how these aspects should be defined by the tester, and will provide an example in the next section. These elements are orchestrated by the loop shown in Algorithm 1, which takes as inputs the target state $s_g$. Furthermore, the tester can choose a search algorithm to update the command sequence $S$, and a tolerance value $\epsilon$ that represents the stop condition of the search algorithm.

**Algorithm 1** WGMainLoop($s_g$,$\epsilon$)

```
1: while D(s_g) − D(s_c) ≥ ε and T ≤ MAX_TIME do
2:     update the action sequence S
3:     apply the last action in sequence S on the SUT
4:     update the current state s_c
5: end while
```

### A. State

The problem of the state definition is of paramount importance in our framework, and it is strictly dependent on the SUT. The framework is aimed at complex and "black-box" systems, for which a detailed knowledge about its internals is not available. Therefore, we consider as "state" a vector of *variables* that reflect the state of a subset *resources* or *data structures* in the system that are relevant for the dependability and performance of the SUT. Since several studies on field failure data have shown that fault activation and propagation is influenced by stress conditions [23], [24], the definition of state may include the usage of internal resources such as buffers, queues and communication channels, and performance measures such as the throughput of the system [25]. Moreover, the state can be defined based on the objectives of the fault injection and from the requirements of the system. For instance, in [9] fault injection is adopted in the context of a fault-tolerant distributed system based on a warm-replication mechanism to copy the state of a process to a backup replica: in this scenario, the evaluation of fault tolerance takes into account the *number of requests in the queues of the process*, which affects the amount of data that has to be copied to the backup replica and ultimately on the effectiveness of fault tolerance, and it is included in the state.

### B. Actions

In general terms, actions are changes in the workload generated by the WG. They may consist in individual messages or commands sent to the SUT, or may represent parameters of a synthetic workload (as in the case of FTAPE [11]). The choice of the set of actions to adopt is dependent on the SUT, since the workload exercises the system through its interface to users and to other systems. Moreover, the definition of actions is also affected by the state definition, since the actions should be able to modify the state of the SUT. For instance, in the case of a web server, actions may change of the rate and kind of HTTP requests [26].

### C. Feedback function

The WG selects actions to be performed on the basis of the feedback from the SUT, by means of the feedback function $D$. This function embeds the fact that a command that drives the system in a state $s_t$ close to $s_g$ gives a positive feedback; the function assumes its maximum value in $s_t = s_g$. The function should compute a scalar value that could be analyzed by the search algorithm, by accounting for the current value of state variables and their target value. For instance, if the WG aims to maximize the throughput of the system during the test, the function could return the current throughput. If more than one state variable is involved, the feedback function could compute a weighted sum of the factors that have to be tuned by the WG.

### D. Search Algorithms

The core of the WG is based on search algorithms. In fact, the workload generation problem is an optimization problem in a discrete space, since the action space is typically discrete. It is known that this problem is NP-hard, therefore the search algorithm should be based on some kind of heuristic. The choice for the action to perform is based on the values returned

 ISBN: 978-1-61208-212-7

by the $D$ function. If the last actions provided an increase of the $D$ function (e.g., $D(s_t) - D(s_{t-1}) > 0$ for the action performed at time $t$), then the algorithm should take into account those actions to build the action sequence S; otherwise, the search algorithm could consider to discard that action in the sequence, and to try a new action. The tester can adopt well-known algorithms in the fields of combinatorial optimization and artificial intelligence, such as simulated annealing, genetic algorithms, and A*.

## V. Evaluation

In this section we illustrate the proposed workload generation framework in the context of a case study and provide a preliminary evaluation. We consider a scenario of a fault injection campaign targeting a Linux-based system, in which experiments have to be performed when the CPU and I/O usage (which represents the state of the system) is equal to a given value. CPU and I/O are exercised by a synthetic workload consisting of CPU-bound and I/O-bound processes, which stress the system through the OS interface. Several fault injection studies were conducted in scenarios similar to ours [27], [11], [12]. In order to control CPU and I/O usage, the WG instantiates CPU-bound and I/O-bound processes, and tunes the number and type of processes using a search algorithm (see Section IV). We evaluate the ability of the proposed framework to find the best mix of CPU-bound and I/O-processes for reaching a desired level of CPU and I/O usage.

### A. System under test

The SUT consists of the Linux OS running in a quad-cpu system. The state of this system (the average utilization of CPU and I/O) can be controlled by spawning synthetic processes that perform CPU- and I/O-intensive operations. However, it is tricky to tune the relative amount of CPU and I/O operations that brings the average CPU and I/O usage to a desired level, since there is a mutual relationship between CPU and I/O operations. In fact, I/O-bound processes often require to use the CPU for short time periods, in order to prepare I/O commands and to send or to retrieve data; therefore, CPU-bound processes may delay I/O-bound processes and affect I/O usage, and I/O-bound processes contribute to CPU usage. CPU and I/O usage is mainly affected by the *process scheduler* of the OS, which selects the order in which CPU- and I/O-bound processes are executed.

In order to evaluate the proposed framework, we adopted *Linsched* [28], a simulator of the Linux OS in multi-cpu environments. It is important to note that Linsched is based on the *actual source code* of the Linux kernel (v. 2.6.35), and that it allows to simulate process execution with high accuracy. In particular, Linsched includes the whole source code of the Linux process scheduler, namely the *Completely Fair Scheduler* (CFS) [29], and allows to evaluate the effect of process scheduling on CPU and I/O usage. Fig. 2 shows the scenario considered in our experiments, which consists of 4 CPUs and an I/O device each associated with a process queue. The system state is defined by two variables, that is, *the average number of processes in CPU queues (runqueues) and in the I/O queue* respectively. In order to control the system state, the WG generates CPU bound and I/O-bound processes, which are allocated to a CPU by a load-balancing algorithm in the Linux kernel, and each CPU is managed using the CFS algorithm. When a process performs an I/O operation, it is moved in the I/O queue, which is managed using a First-In-First-Out algorithm. The complex relationship between CPU and I/O usage can be seen in Fig. 3, which shows the average length of each queue as a function of the number of CPU bound and I/O-bound processes: these functions are non-linear, and where one of the functions increases, the other one decreases.

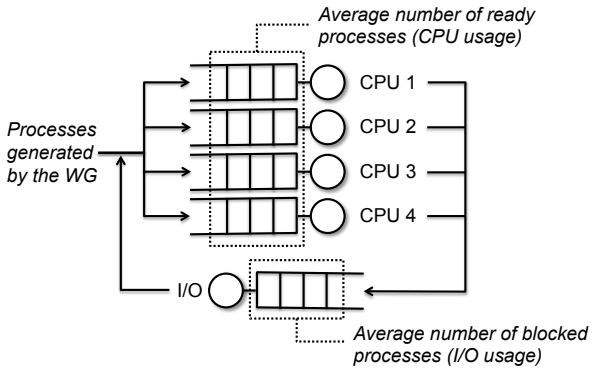


Fig. 2: Execution scenario.

### B. Instantiating the WG framework

Following the proposed framework, we have defined the state $s$ as a vector of two components: (i) the average number of the processes in the runqueues (ii) the average number of processes in I/O queue. The function $D$ is defined as:

$$D(s_t) = -||s_t - s_g||$$

where $||.||$ is the euclidean norm between the state vectors $s_t$ and $s_g$. The processes that can be generated by the WG are of three types, as shown in TABLE I. The first column is the process type, that can be CPU-bound, I/O-bound or Mixed; the second and the third column provide the average time that a process of each type spends in the *running state* (i.e., it is using the CPU) and in the *blocked state* (i.e., it is waiting for the completion of an I/O operation), respectively. The WG starts its execution using a process set filled with $n_1$ CPU-bound processes, $n_2$ I/O-bound processes, and $n_3$ Mixed processes (we assume $n_1 = n_2 = n_3 = 10$ as initial value) and we have set $\epsilon$ to $0.4$. The actions of the WG consist in increasing or decreasing (by one, five, or ten) the number of processes of each type ($n_1$, $n_2$, and/or $n_3$). For every action $a$ in the action set $A$, we refer to the action opposite to $a$ as $a'$, which removes the effects of $a$. In the case that such actions are not available or applicable for the target system, the WG should at each iteration (i) reset the system state, and (ii) apply the whole sequence $S$. Finally, we have chosen the Simulated

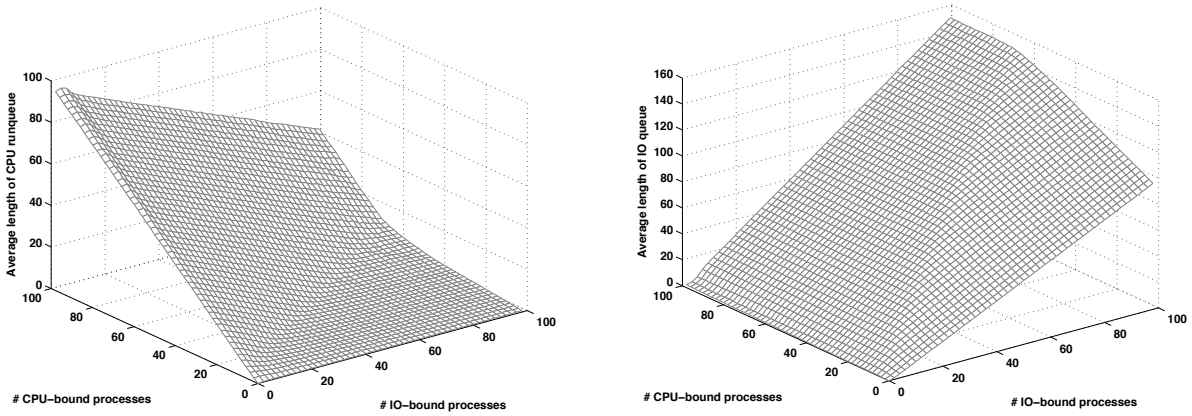


Fig. 3: Average length of CPU and I/O queues in function of the number of CPU- and I/O-bound processes in the system.

Annealing [30] as search algorithm (Algorithm 2), which is executed in the context of the WG main loop (Algorithm 1). It randomly chooses an action, and evaluates $D$ in the new state $s_n$. If the action produced an increase of $D$ compared to the previous state, then the action is appended to the action sequence. If $D$ decreases, then the action is not included in the sequence and the system is reverted to its previous state using the action $a'$. However, the algorithm may still include action $a$ in the sequence and remain in the state $s_n$ (even if $D$ decreased) with probability $e^{-\Delta E/T}$ (decreasing with time).

**Algorithm 2** SIMULATEDANNEALING($s_c$)

1: $a \leftarrow random\_select(A)$
2: do $a$ on SUT and get the next state $s_n$
3: $\Delta E \leftarrow R(s_n) \leftarrow D(s_n) - D(s_c)$
4: **if** $\Delta E < 0$ **and** $random(1 - e^{-\Delta E/T})$ **then**
5: do $a'$ on SUT
6: **end if**

TABLE I: Process types instantiated by the WG.

| Type | Mean CPU time | Mean I/O time |
|---|---|---|
| CPU-bound processes | 200 ticks | 20 ticks |
| I/O-bound processes | 5 ticks | 20 ticks |
| Mixed processes | 20 ticks | 20 ticks |

### C. Experimental results

In our experiments, we evaluated the framework using 9 different goal states, by selecting different realistic conditions in which the SUT could be operating. The target values for the average length of CPU queues were respectively $low = 10$, $medium = 50$, and $high = 100$. In a similar way, the target values for the average length of the I/O queue were respectively $low = 10$, $medium = 50$, and $high = 100$. At every iteration of Algorithm 1, the system is simulated for 10 seconds, and the state of the system is collected by computing the average number of ready and blocked processes. We evaluate the number of iterations for reaching the goal state.

In every experiment, the WG reached the target state. The distance from the target state approaches to zero in at most 250 iterations, as shown in Fig. 4. The worst case is represented by the target state $(100, 100)$, which is the state farther from the initial state (i.e., the state obtained using the initial mix of processes $n_1$, $n_2$, and $n_3$). It could possible to improve the speed of convergence using a different initial mix of processes. For instance, a tester could consider a set of several random process mixes, and the Workload Generator can choose as initial mix the one that is closer in the state space to the target state. It is important to note that this is a general problem of search and optimization algorithms. Figure 5 shows how the search algorithm moves towards the target state. The oscillations are due to the random choices made by the search algorithm, which in the long term brings the system in the target state (the filled dot in the figure).

## VI. CONCLUSION

In this paper, we discussed a framework for automatically generating a workload to reach a target state defined by the tester. In this framework, a Workload Generator interacts with the System Under Test in a closed loop, to iteratively search for a sequence of actions that brings the System Under Test in the target state. The framework has been evaluated in a real complex system, namely the Linux 2.6 OS scheduler. From the results, we found that the Workload Generator is able to reach the target state in every experiment in a reasonable number of iterations. Future work in this area can be focused on the application of the approach to other systems, with the aim of evaluating and improving its effectiveness and portability. Another area worth of investigation is the adoption of the framework in the context of dependability benchmarking.

## ACKNOWLEDGMENT

This work has been supported by the projects "Embedded Systems in Critical Domains" (CUP B25B09000100007)

 ISBN: 978-1-61208-212-7

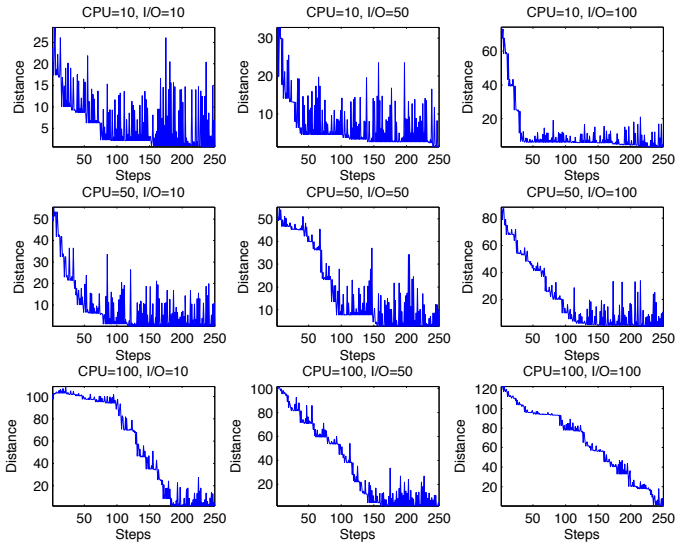


Fig. 4: Speed of convergence of the Workload Generator.

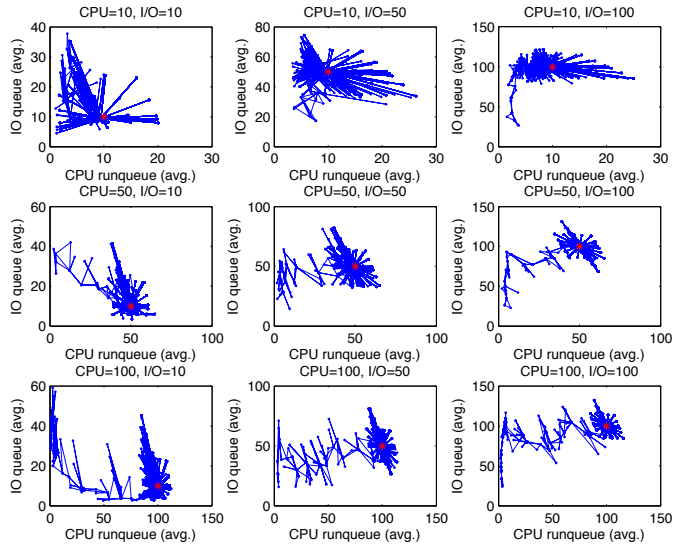


Fig. 5: State space explored by the Workload Generator.

within the framework of ”POR Campania FSE 2007-2013” and ”Iniziativa Software CINI-Finmeccanica”.

## REFERENCES

[1] R. Lefever, M. Cukier, and W. Sanders, “An experimental evaluation of correlated network partitions in the Coda distributed file system,” in *Proc. Intl. Symp. Reliable Distributed Systems*, 2003, pp. 273–282.

[2] M. Vieira and H. Madeira, “A dependability benchmark for OLTP application environments,” in *Proc. 29th Intl. Conf. on Very Large Data Bases*, 2003, pp. 742–753.

[3] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. Fabre, J. Laprie, E. Martins, and D. Powell, “Fault injection for dependability validation: A methodology and some applications,” *IEEE Trans. Software Eng.*, vol. 16, no. 2, pp. 166–182, 1990.

[4] K. Joshi, M. Cukier, and W. Sanders, “Experimental evaluation of the unavailability induced by a group membership protocol,” *Proc. European Dependable Computing Conf.*, pp. 644–648, 2002.

[5] B. Helvik, H. Meling, and A. Montresor, “An approach to experimentally obtain service dependability characteristics of the Jgroup/ARM system,” *Proc. European Dependable Computing Conf.*, pp. 179–198, 2005.

[6] E. Czeck and D. Siewiorek, “Observations on the Effects of Fault Manifestation as a Function of Workload,” *IEEE Trans. Computers*, vol. 41, no. 5, pp. 559–566, 1992.

[7] R. Chillarege and R. Iyer, “An experimental study of memory fault latency,” *IEEE Trans. Computers*, vol. 38, no. 6, pp. 869–874, 1989.

[8] J. Meyer and L. Wei, “Analysis of workload influence on dependability,” in *Proc. Intl. Fault-Tolerant Computing Symp.*, 1988, pp. 84–89.

[9] R. Natella and D. Cotroneo, “Emulation of transient software faults for dependability assessment: A case study,” in *Proc. European Dependable Computing Conf.*, 2010, pp. 23–32.

[10] R. Chandra, R. Lefever, K. Joshi, M. Cukier, and W. Sanders, “A Global-State-Triggered Fault Injector for Distributed System Evaluation,” *IEEE Trans. Parallel and Distributed Sys.*, vol. 15, no. 7, pp. 593–605, 2004.

[11] T. Tsai and R. Iyer, “Measuring fault tolerance with the FTAPE fault injection tool,” *Quantitative Evaluation of Computing and Communication Systems*, pp. 26–40, 1995.

[12] T. Tsai, M. Hsueh, H. Zhao, Z. Kalbarczyk, and R. Iyer, “Stress-Based and Path-Based Fault Injection,” *IEEE Trans. Computers*, vol. 48, no. 11, pp. 1183–1201, 1999.

[13] S. Dawson, F. Jahanian, T. Mitton, and T. Tung, “Testing of fault-tolerant and real-time distributed systems via protocol fault injection,” in *Proc. Fault Tolerant Computing Symp.*, 1996, pp. 404–414.

[14] D. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. Iyer, “Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors,” in *Proc. IEEE Intl. Computer Performance and Dependability Symp.*, 2000, pp. 91–100.

[15] W. Hoarau and S. Tixeuil, “A language-driven tool for fault injection in distributed systems,” in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, 2005, pp. 194–201.

[16] D. Avresky, J. Arlat, J. Laprie, and Y. Crouzet, “Fault Injection for Formal Testing of Fault Tolerance,” *IEEE Trans. Reliability*, vol. 45, no. 3, pp. 443–455, 1996.

[17] A. Arazo and Y. Crouzet, “Formal Guides for Experimentally Verifying Complex Software-Implemented Fault Tolerance Mechanisms,” in *Proc. Intl. Conf. on Eng. of Complex Computer Systems*, 2001, pp. 69–79.

[18] A. Ambrosio, F. Mattiello-Francisco, V. Santiago, W. Silva, and E. Martins, “Designing Fault Injection Experiments Using State-Based Model to Test a Space Software,” *Lecture Notes in Comp. Science*, vol. 4746, pp. 170–178, 2007.

[19] A. Johansson, N. Suri, and B. Murphy, “On the Impact of Injection Triggers for OS Robustness Evaluation,” in *The 18th Intl. Symp. on Software Reliability Eng.*, 2007, pp. 127–136.

[20] S. Dawson, F. Jahanian, and T. Mitton, “Experiments on six commercial TCP implementations using a software fault injection tool,” *Software Practice and Experience*, vol. 27, no. 12, pp. 1385–1410, 1997.

[21] V. Prabhakaran, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, “Model-based failure analysis of journaling file systems,” in *Proc. Intl. Conf. Dependable Systems and Networks*, 2005, pp. 802–811.

[22] D. Cotroneo, D. Di Leo, R. Natella, and R. Pietrantuono, “A case study on state-based robustness testing of an operating system for the avionic domain,” *Lecture Notes in Comp. Science*, vol. 6894, pp. 213–227, 2011.

[23] M. Sullivan and R. Chillarege, “Software Defects and their Impact on System Availability: A Study of Field Failures in Operating Systems,” in *Proc. Intl. Fault-Tolerant Computing Symp.*, 1991, pp. 2–9.

[24] I. Lee and R. Iyer, “Software dependability in the Tandem GUARDIAN system,” *IEEE Trans. Software Eng.*, vol. 21, no. 5, pp. 455–467, 1995.

[25] R. Jain, *The art of computer systems performance analysis*. John Wiley & Sons, 2008.

[26] R. Matias *et al.*, “An experimental study on software aging and rejuvenation in web servers,” in *Proc. Computer Software and Applications Conf.*, vol. 1, 2006, pp. 189–196.

[27] W.-I. Kao, R. Iyer, and D. Tang, “FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behavior under Faults,” *IEEE Trans. Software Eng.*, vol. 19, no. 11, pp. 1105–1118, 1993.

[28] J. Calandrino, D. Baumberger, T. Li, J. Young, and S. Hahn, “Linsched: The linux scheduler simulator,” in *Proc. Intl. Conf. on Parallel and Distributed Comp. and Comm. Systems*, 2008, pp. 171–176.

[29] C. Wong, I. Tan, R. Kumari, and F. Wey, “Towards achieving fairness in the linux scheduler,” *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 34–43, 2008.

[30] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, pp. 671–680, 1983.

 ISBN: 978-1-61208-212-7

# Improving Consumer Satisfaction through Building an Allocation Cloud

Kuo Chen Wu
*Institute of Computer and Communication Engineering*
*National Cheng Kung University*
*Tainan, Taiwan*
*kuchenwu@nature.ee.ncku.edu.tw*

Hewijin Christine Jiau, Kuo-Feng Ssu
*Department of Electrical Engineering*
*National Cheng Kung University*
*Tainan, Taiwan*
{*jiauhjc, ssu*}*@mail.ncku.edu.tw*

***Abstract*—Consumer satisfaction depends on the quality of service. Services with high quality usually lead to high cost. Due to the budget limitation, consumers tend to acquire services with lowest cost. Feasible request allocation can improve service quality or reduce service cost. Some request allocation algorithms need the coordination with service consumers, but the benefits from consumers' coordination are arranged by service providers, especially for on-demand consumers. On-demand consumers have no idea to improve service quality or reduce service cost through allocating their requests, even though service providers can collect on-demand requests to perform feasible request allocation for efficient service execution. To solve this issue, an allocation cloud is proposed in this study. The allocation cloud assists consumers in allocating requests to optimize consumer satisfaction. Because the allocation cloud contains various allocation algorithms and collects a large amount of service requests, the allocation cloud can coordinate service consumers to follow feasible request allocation algorithms. Moreover, the allocation cloud can reveal the relation between service consumers' coordination and the improvement of service execution. The revealed relation can help service consumers decide beneficial strategies to improve their satisfaction. To evaluate the effectiveness of our approach, a study of video delivery service is conducted. The results confirm that consumer satisfaction can be improved through the proposed allocation cloud.**

***Keywords-Cloud Computing; Quality of Service; Request Allocation; Consumer Satisfaction.***

## I. INTRODUCTION

High quality of service (QoS) leads to high consumer satisfaction and also means high service cost. For example, service respond time increasing usually needs to increase network bandwidth, computation power, or cache size. The addition of resources means the cost increasing. However, service consumers always has budget limitation and need to consider the trade-off between QoS improvement and service cost increasing. Besides considering the trade-off between QoS and service cost, consumers can try to reduce through two approaches. One approach is making a favourable service agreement with service providers and the other approach is performing service sharing through appropriate request allocation. However, On-Demand consumers ask services when they need and usually send small numbers of requests. Their characteristics make the cost reduction through request allocation or service agreement becomes difficult. The reasons are listed as follows:

*1)* The amount of service requests from each On-Demand consumer is small. It is hard for on-demand consumers to gain benefits by performing request allocation with a small amount of requests.
*2)* An on-demand consumer cannot guarantee the amount of service requests. Therefore, service providers do not make a favourable service agreement with an On-Demand consumer.

Therefore, it is difficult for On-Demand consumers alone to improve consumer satisfaction through request allocation and service agreements. However, via collecting the requests from on-demand consumers, service providers can still perform suitable request allocation for efficient service execution. Through the suitable request allocation, on-demand consumers can acquire efficient service execution. However, on-demand consumers have no idea to coordinate with other on-demand consumers to follow feasible requestion allocation and gain additional benefits.

We propose an allocation cloud to coordinate on-demand consumers for efficient request allocation. The allocation cloud contains two important concepts. The first concept is the collection of on-demand consumers' requests. The allocation cloud coordinates on-demand consumers to follow request allocation according to these on-demand consumers' requests. The requests contain the functional and non-functional requirements of requesting services. The non-functional requirements are presented as hard constraints and soft constraints [1] in this study. Hard and soft constraints are criteria to evaluate request allocation in this study. Furthermore, on-demand consumers' coordination and request allocation are based on the evaluation of hard and soft constraints. The second concept is the request allocation algorithm set. A request allocation algorithm specifies targeted request collection, performs request allocation for the request collection, and presents the effects of applying the request allocation. Multiple allocation algorithms are available in the allocation cloud for different request collections. With these allocation algorithms, the allocation cloud can coordinate consumers to follow feasible allocation

 ISBN: 978-1-61208-212-7

algorithms for corresponding request collection. In addition, the allocation cloud can further improve service execution using how consumers' coordination for more efficient resource usage or better QoS. Keeping the relation between consumers' coordination and service execution is important for making service agreements with service providers. On-demand consumers and the allocation cloud can be aware that their coordination really contributes the service quality improvement. Therefore, the allocation cloud makes service agreements with service providers for improving QoS or reducing prices.

This paper is organized as follows: Section II lists relevant studies which discuss the relation between consumer satisfaction and cloud services. Section III introduces the major concept of the allocation cloud. Section IV discusses the effectiveness of the allocation cloud through an experiment. The experiment which simulates the video delivery service with *Coordinated Channel Allocation* [2] (COCA) is attached. Section V is the conclusion of this paper.

## II. Related Work

Due to increasing popularity of cloud computing [3][4], more and more service providers utilize clouds as their deployment platforms [5][6][7]. There are sufficient resources for service executing in cloud platforms, and how to allocate service requests adequately and optimize profit [8] or consumer satisfaction becomes important.

Several studies define and explore consumer satisfaction of using cloud services. Tsakalozos et al. [9] explore the consideration of *Infrastructure as a Service* (*IaaS*) cloud administrator to maximize cloud-consumer satisfaction. They map cloud-consumer satisfaction into financial profit, which is defined as the difference between revenue and cost. Both of revenue and cost are impacted by the cloud's physical resource usage. Besides that, there are two assumptions in this study. The first assumption is that a cloud-consumer has a budget constraint. The second one is that the cloud's physical resources are limited. They develop the approach for cloud administrators to maximize per-consumer financial profit through allocating physical resources.

Kantere et al. [10] propose an pricing method to optimize the profit of cloud cache services. They expect that consumer dissatisfaction from high service charge leads to the dropping of service demand. Besides that, the cloud risks to permanently lose the dissatisfied consumers. Base on this expectation, consumer satisfaction is the factor of optimizing profit. The consumer satisfaction is an altruistic tend of the optimization that is opposite to the egoistic tend of cloud profit. They express altruistic tend as: 1) a guarantee for a low limit on consumer satisfaction, or 2) an additional maximization objective. According to the expression, they optimize cloud cache service profit with consider consumer satisfaction.

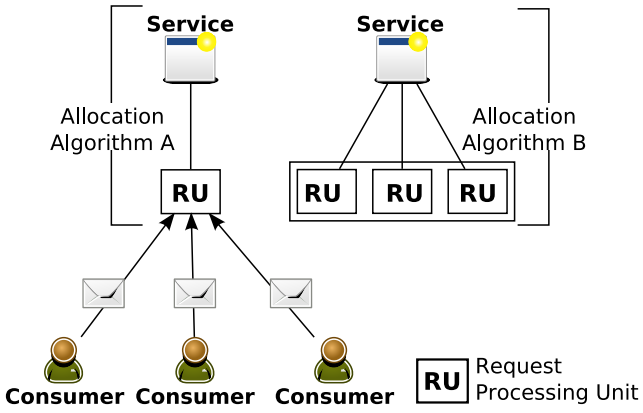


Figure 1. Request allocation without the allocation cloud.

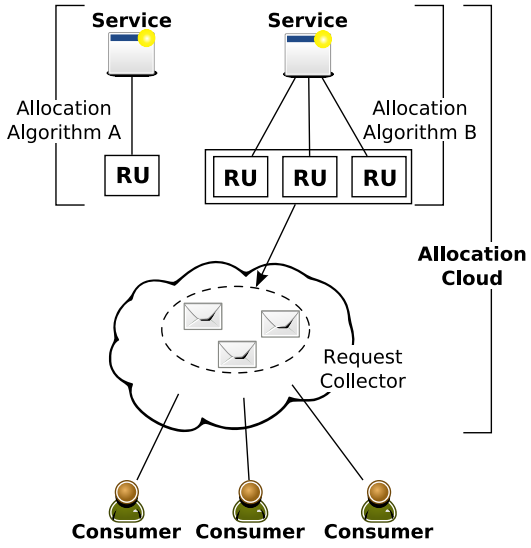


Figure 2. Request allocation within the allocation cloud.

Chen et al. [11] develop an utility model for measuring consumer satisfaction in the cloud. Since they focus on the service provisioning problem, the utility model defines consumer satisfaction with two factors: service price and response time. They assume that consumer satisfaction is decreased with higher service price and longer response time. The developed utility model can help service providers build services in the *IaaS* cloud platform with maximal profit.

## III. Allocation Cloud

Consumers want to perform efficient request allocation to share services and reduce cost. However, consumers are limited to perform efficient request allocation because they ask for services independently and usually have a small amount of requests. Figure 1 illustrates such situation. In Figure 1, each consumer has one request to the same service. The desired service provides two allocation algorithms. Both the algorithm A and the algorithm B reserve request processing

 ISBN: 978-1-61208-212-7

units to serve request. The algorithm B allows three the same requests to share the service. However, consumers can only perform request allocation with allocation algorithm A since each consumer does not know they can cooperate to perform allocation algorithm B. The allocation cloud is proposed to solve this issue. Figure 2 illustrates the overview of the allocation cloud. In Figure 2, the allocation cloud includes several allocation algorithms and a request collector. The allocation cloud collects consumers' requests and reserves them in the request collector. The allocation cloud can find several requests that can be shared with the same service because the allocation cloud can evaluate the global requesting situation. The allocation cloud subsequently performs request allocation with allocation algorithm B. As a result of service sharing, consumers can reduce cost and improve satisfaction through the allocation cloud.

Besides the request allocation, there are two operations in the allocation cloud which can improve consumer satisfaction. The first operation is the service agreement through the allocation cloud. Because the allocation cloud reserves requests, the allocation cloud can predict service requesting situation and make service agreements according to the prediction. While the actual service requesting fulfils the prediction, the cost can be reduced and consumer satisfaction can be improved. The second operation is the consumer negotiation in the allocation cloud. The allocation cloud can negotiate service requesting with consumers when actual service requesting is not conformed to the expected situation. If consumers accept to modify their requesting to fulfil the expectation, consumer satisfaction can be improved.

In the following sections, the definition of consumer satisfaction is described in Section III-A. Operations of the allocation cloud, request allocation, service agreement, and consumer negotiation, are explained in Section III-B, III-C and III-D, respectively. Finally, Section III-E discusses the restriction of building an allocation cloud.

### A. Consumer Satisfaction

Consumer satisfaction in the allocation cloud is the sum of all request satisfaction. Given all requests $R = \{r_1, \ldots, r_m\}$ in the cloud, every request satisfaction is impacted by two factors. One factor is the evaluation on the quality $q_r$ that the service exhibits and the other factor is the service price. $QE_r$ is the function of quality evaluation for request $r$ and is specified by hard constraints, $C_{Hr}$, and soft constraints, $C_{Sr}$. The quality evaluation $QE_r$ on $q_r$ is presented as follows.

$$QE_r(q_r) = C_{Hr}(q_r) \cdot C_{Sr}(q_r), where \quad (1)$$

$$\begin{cases} C_{Hr}(q_r) & \in & \{0,1\} \\ C_{Sr}(q_r) & = & [0.0, 1.0] \end{cases}$$

If $q_r$ does not conform to $C_{Hr}$, the value of $C_{Hr}(q_r)$ is 0 that means that $q_r$ is not satisfied with hard constraints. Therefore, $q_r$ must conform to $C_{Hr}$ first and the influence of $C_{Sr}(q_r)$ is revealed. Based on $QE_r$, the request satisfaction $S_r$ is presented as follows.

$$S_r(q_r) = \frac{QE_r(q_r)}{P} \quad (2)$$

The service price is represented as $P$. The low price and the high quality of service lead the high request satisfaction. Based on $S_r$, consumer satisfaction $CS$ in the allocation cloud is presented as follows.

$$CS = \frac{\sum_{r \in R} S_r(q_r)}{\mid R \mid} \quad (3)$$

The $CS$ function is defined as the ratio of total request satisfaction values to the total request amount in the allocation cloud. It should be noted that $q_r$ and $P$ exhibited by a service vary upon different service types. Also, the specifications of $C_{Hr}$ and $C_{Sr}$ are related to service requests. Based on $CS$ function, the allocation cloud can improve consumer satisfaction by performing optimization techniques. The cloud operations on request allocation, service agreement, and consumer negotiation can be executed to achieve the optimization.

### B. Request Allocation

The allocation cloud can perform alternative allocation algorithms. Different allocation algorithms lead to a change of service quality and service cost. The allocation cloud needs to estimate the change of $q_r$ and $P$ and attempts to optimize consumer satisfaction defined in (3) from available allocation algorithms. Besides, there are some rules that should be fulfilled before allocating requests, as listed as follows.

$$\forall r \in R, C_{Hr}(q_r) \leq C_{Hr}(q'_r) \quad (4)$$

$$\sum_{r \in R} QE_r(q_r) \leq \sum_{r \in R} QE_r(q'_r) \quad (5)$$

In (4) and (5), $q'_r$ is the service quality with alternative allocation algorithms. Equation (4) ensures that $q'_r$ cannot be worse that $q_r$. If $q_r$ conforms to $C_{Hr}$, $q'_r$ must also conform to $C_{Hr}$. The allocation cloud needs to make sure that all involved consumers are not sacrificed in alternative allocation algorithms. Equation (5) ensures that $\sum_{r \in R} QE_r(q_r)$ cannot be worse within alternative allocation algorithms. The allocation cloud needs to promise that QoS of all requests are not sacrificed in alternative allocation algorithms.

### C. Service Agreement

Request allocation in the allocation cloud can lead request fulfilment with lowest cost under the prerequisite of (4) and (5). Besides, the allocation cloud has to prepare enough services for requests with expected quality constraints. The allocation cloud, instead of consumers, performs service agreements with service providers to prepare enough services. Figure 3 illustrates the communication protocol for

 ISBN: 978-1-61208-212-7

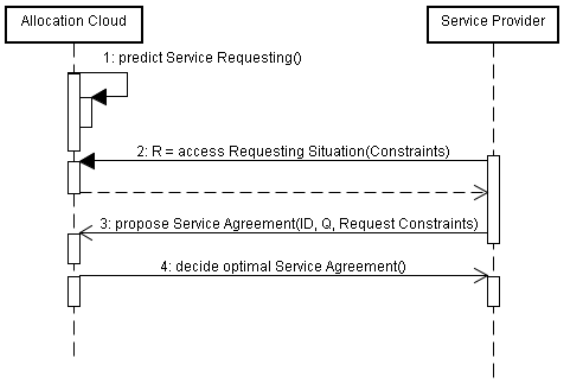


Figure 3. A service agreement protocol.

making service agreements between the allocation cloud and service providers. As shown in Figure 3, the allocation cloud does not find suitable services through service brokers and does not actively contact with service providers. Alternatively, service providers send query messages to the allocation cloud to check the service requesting situation. After confirming the service requesting situation, service providers can propose suitable service agreements according to this situation.

The communication protocol has four steps, as follows

1) *predict Service Requesting():* The allocation cloud can collect historical service requesting records. Therefore, the allocation cloud predicts service requesting according to historical service requesting records and all other related information. The prediction is represented as $R = \{r_1, \ldots, r_m\}$ and $\forall r \in R$ includes $C_{Hr}$ and $C_{Sr}$
2) *access Requesting Situation(Query Statement):* Service providers query service requesting situation in this step. Service providers can specify query messages, such as needed service amount or occupied time. Service providers can concentrate on target service requesting through specifying query messages. The allocation cloud receives the query messages and returns service requesting situation according to the query messages.
3) *propose Service Agreement(ID, Q, Request Constraints):* Service providers can decide whether to propose service agreements or not according to the received service requesting situation. If service providers judge that they can propose profitable service agreements, they continue to propose service agreements to the allocation cloud. The proposed service agreements need to contain three parameters, `ID`, `Q`, and `Request Constraints`. The `ID` is the identification of this service agreement and is used by the allocation cloud to recognize this service agreement. The `Q` is QoS for this service agreement. The allocation cloud uses this parameter to judge consumer satisfaction of this service agreement. The `Request Constraints` are the constraints which the allocated requests must be conformed to. Service providers use this parameter to specify target service requesting.
4) *decide optimal Service Agreements():* The purpose of step 4 is to select a set of service agreements from all received service agreements. This selected set of service agreements can optimize consumer satisfaction in the allocation cloud. The fitness function of the optimization is the $CS$ function defined in (3). The allocation cloud simulates the request allocation with different allocation algorithms and calculates the value of $CS$. The allocation cloud returns selected service providers to achieve service agreements when deciding the set of service agreements which can optimize consumer satisfaction.

This communication protocol is proposed for the listed reasons.

- The allocation cloud can trigger service providers to propose service agreements actively because the allocation cloud keeps a large amount of service requests.
- Service providers can be aware of the service requesting situation through the allocation cloud. Service providers can propose the most profitable service agreements specialized for target service requesting.
- The allocation cloud does not need to specify service specification and find suitable services. The allocation cloud can concentrate on comparing proposed service agreements and on deciding the set of service agreements which lead to consumer satisfaction optimization.
- Because service providers only propose their service agreements for target service requesting, the amount of proposed service agreements is less than the amount of all service agreements which service providers can provide. This situation reduces the complexity of achieving service agreements

### D. Consumer Negotiation

The allocation cloud makes service agreements based on the expected service requesting situation. Consumer satisfaction optimization is achieved in accordance with the expected situation. However, actual service requesting situation is different with the expected situation and consumer satisfaction is not as good as the expectation. Consumer satisfaction can be improved when the actual service requests are adjusted to fulfil the expected situation. For this purpose, the allocation cloud negotiates quality constraint refinement of service requests with consumers. The allocation cloud negotiates with consumers who send requests fulfilled the conditions, as listed as follows.

- The service request $r \in R$ proposed by a consumer is similar to the expected service request for a specific service.
- Request satisfaction $S_r$ will be improved if the allocation cloud modifies quality constraints of this request $r$.

 ISBN: 978-1-61208-212-7 

- Consumer satisfaction $CS$ in the allocation cloud will also be improved if the allocation cloud modifies quality constraints of this request $r$.

The purpose of the allocation cloud is to serve consumers and improve consumer satisfaction. Therefore, the allocation cloud must allow consumers to be aware of the modification of quality constraints and request satisfaction.

### *E. Restrictions*

The major restriction of building an allocation cloud are privacy and scalability concerns. The administrator of the allocation cloud must be trusted by target consumers because the allocation cloud collects service requests from consumers. Besides that, the allocation cloud must collect enough requests in order to trigger service providers to propose their service agreements for the allocation cloud.

## IV. EXPERIMENT

An experiment is conducted to evaluate the effectiveness of the allocation cloud. In this experiment, streaming video delivery service is selected as the targeted application. Due to the high resource need and heavy workload, selectively feasible allocation algorithm is important to guarantee system capability with efficient resource utilization. Some allocation algorithms need to coordinate with service consumers, such as *coordinated channel allocation* [2] (explained in the next section). However, the benefits of request allocation algorithms only reflects on the service providers, even though the request allocation algorithms need consumers' coordination. We regard these allocation algorithms as services in the allocation cloud for service consumers and evaluate the improvement of consumer satisfaction using the allocation cloud. There are two assumptions for resolving the concerns for the restrictions mentioned in Section III-E. The first assumption is that consumers of the video delivery service have minor privacy concerns and allow the allocation cloud to allocate their video requests. The second assumption is that the allocation cloud can collect large amount of requests. We make the second assumption because streaming video is popular and most consumers request video on-demand. In the following contents, coordinated channel allocation is introduced. The benefits of the allocation cloud is explained while applying COCA to the allocation cloud.

### *A. Coordinated Channel Allocation for Video Delivery Services*

COCA is based on batching techniques [12][13][14]. Therefore, these requests are grouped together in a small interval. A scheduler with COCA negotiates with consumers for reserving the requests with larger resource needs and sets a deadline of starting service for each reserved group. Based on the reserved groups, the scheduler with COCA has two queues, reservation queue and regular queue. The reservation queue keeps all reserving groups and performs sort on groups according to group deadline. The regular queue keeps all awaiting groups and performs sort on all groups according to group arrival time and resource needs. When there exists free channels and the deadline of the group in the reservation queue is reached, the scheduler with COCA allocate the group in the reservation queue to a free channel. When there exists free channels but no group deadline is reached, the Scheduler with COCA allocate the group in the regular queue to a free channel. This algorithm avoids that the groups with large resource needs occupy channels for a long time and hinders other groups from using the service.

According to COCA, delivery service providers perform efficient request allocation based on the arrival requests and the coordination with consumers. In other words, if consumers can coordinate with each other before requesting services, consumers can negotiate with service providers for additional benefits. However, there are two reasons make on-demand consumers unable to coordinate with each other. First, on-demand consumers do not know other arrival requests. Since consumers have no idea of other arrival requests, they cannot coordinate with each other. Second, on-demand consumers have no idea about efficient request allocations for the video delivery service. Moreover, video on-demand consumers are usually end users and do not have background knowledge of efficient request allocations. Even if they know other arrival requests, they still cannot coordinate with each other because they have no background knowledge to perform request allocation. Therefore, this case is suitable to apply the allocation cloud. The allocation cloud for video delivery services contains efficient request allocation algorithms and regard these algorithms as services. The allocation cloud collects the information of video requests and available video delivery services. Through actively detecting the video requests and the available video delivery services, the allocation cloud can decide feasible allocation services, such as COCA, and negotiate with consumers for request allocation. Since the allocation cloud can be aware of how consumer coordination impacts the profit of service providers, it can negotiate with service providers for consumers' addition benefits, such as video discount.

### *B. The Relation between Consumer Coordination and Service Providers' Profit*

**Experiment Setting:** It is assumed that a video delivery service owns 500 videos and the average delivery time of a video is 30 minutes. The allocation cloud predicts the service requesting situation in the next one hour. The request arrival rate is from 5 to 30 requests per minute and is based on a Poisson distribution. The maximum waiting period of each request is between 2.5 minutes and 7.5 minutes and is based on an exponential distribution. It is assumed that there are four available allocation services,

 ISBN: 978-1-61208-212-7

Table I
COMPARISON THE DIFFERENCE OF RENEGE RATE FOR STRATEGIES WITHOUT COCA AND WITH COCA IN THE SIMULATION RESULT. RENEGE RATE IS THE NUMBER OF RENEGED REQUESTS DIVIDED BY THE TOTAL NUMBER OF REQUESTS MADE IN THE EXPERIMENT.

| *Channels* | On-Demand | | | PRC | | |
|---|---|---|---|---|---|---|
| | Without COCA | With COCA | Difference | Without COCA | With COCA | Difference |
| 100 | 60% | 34% | 26% | 56% | 29% | 27% |
| 120 | 55% | 34% | 21% | 51% | 27% | 24% |
| 140 | 52% | 30% | 22% | 46% | 26% | 20% |
| 160 | 46% | 29% | 17% | 42% | 24% | 18% |
| 180 | 41% | 26% | 15% | 37% | 23% | 14% |

Table II
COMPARISON THE IMPROVEMENT OF SATISFIED REQUESTS FOR STRATEGIES WITHOUT COCA AND WITH COCA IN THE SIMULATION RESULT. SATISFIED REQUESTS IS THE NUMBER OF FULFILLED REQUESTS IN THE EXPERIMENT.

| *Channels* | On-Demand | | | PRC | | |
|---|---|---|---|---|---|---|
| | Without COCA | With COCA | Improvement (%) | Without COCA | With COCA | Improvement (%) |
| 100 | 765 | 1227 | 60.39% | 824 | 1326 | 27% |
| 120 | 844 | 1231 | 45.85% | 907 | 1354 | 24% |
| 140 | 915 | 1301 | 42.50% | 1004 | 1390 | 20% |
| 160 | 1013 | 1330 | 31.29% | 1330 | 1423 | 18% |
| 180 | 1094 | 1375 | 25.69% | 1375 | 1442 | 14% |

On-Demand strategy [14], Pure-Rate-Control (PRC) strategy [15], On-Demand + COCA [2], and PRC + COCA [2]. The allocation cloud simulates the request allocation and the result is shown in Table I and Table II. Table I lists the renege rate in different channel numbers. The renege rate is the ratio of requests finally cancelled by service consumers due to over request deadline. Table II shows the number of satisfied requests in different channel numbers. According to the renege rate and satisfied requests in Table I and II, both on-demand allocation and PRC allocation can improve the number of satisfied requests by applying COCA. The more requests served, the higher profit gained by the service provider. Moreover, the profit is from consumers' coordination. Table I also displays the reduction of renege rate due to consumers' coordination. When video delivery service provides 100 channels, COCA reduces renege rate by 26% on the on-demand strategy and 27% on the PRC strategy. The reduction on the on-demand strategy and the PRC strategy is similar in different number of channels. Table I also shows that the reduction is more notable when the number of available channels is smaller. Table II displays the improvement of satisfied requests due to consumers' coordination. The result shows that satisfied requests can be improved approximately 60% when the video delivery service provides 100 channels. When the video delivery service provides 180 channels, the improvement is still over 20%.

**Applying COCA to the Allocation Cloud:** In brief, the result shows that video delivery services can have better satisfied requests but video requests with larger resource needs have to be reserved until the deadline. This indicates that some requests have to sacrifice their average waiting time to improve the total satisfied requests. Without the allocation cloud, service consumers cannot coordinate with each other and are not aware of their contributions on improving the profit of the video delivery service. With the allocation cloud, these service consumers who sacrifice their average waiting time can be aware of their contributions. The allocation cloud can also make favourable service agreements with video delivery service providers. Therefore, service consumers can reduce service price if they can afford to sacrifice their waiting time.

## V. CONCLUSION AND FUTURE WORK

We propose the allocation cloud to deal with the request allocation for consumers. Consumer satisfaction can be improved through the allocation cloud because it optimizes service utilization and guarantees QoS. We introduce the operations of the allocation cloud and demonstrate its effectiveness through an experiment. According to the experiment results, we conclude that the allocation cloud collects service requests and reveals the relation between consumers' coordination and service execution. Therefore, service consumers can be aware of how to improve their satisfaction through feasible coordination. In the future, the consideration of consumer satisfaction optimization must be further explored. For this purpose, we need to decide a specific service and a set of consumers which make requests for the specific service. According to the specific service and corresponding consumers, allocation algorithms are selected in the allocation cloud. We plan to explore the consumer satisfaction optimization based on these allocation algorithms.

## ACKNOWLEDGMENTS

Part of this work has been supported by National Science Council, Taiwan (NSC 100-2221-E-006-133-MY3, NSC

 ISBN: 978-1-61208-212-7 

100-2628-E-006-028-MY3, and NSC 100-2221-E-006-136-MY2).

## REFERENCES

[1] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[2] Y.-W. Ho, "Coordinated resource allocation for video delivery services," Master's thesis, National Cheng Kung University, Tainan, Taiwan, 2009.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Comm. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[4] M. Creeger, "CTO Roundtable: Cloud computing," *Comm. ACM*, vol. 52, no. 8, pp. 50–56, Aug. 2009.

[5] A. Ojala and P. Tyrväinen, "Developing cloud business models: A case study on cloud gaming," *IEEE Softw.*, vol. 28, no. 4, pp. 42–47, Jul./Aug. 2011.

[6] P. Louridas, "Up in the air: Moving your applications to the cloud," *IEEE Softw.*, vol. 27, no. 4, pp. 6–11, Jul./Aug. 2010.

[7] M. Cusumano, "Cloud computing and SaaS as new computing platforms," *Comm. ACM*, vol. 53, no. 4, pp. 27–29, Apr. 2010.

[8] F. I. Popovici and J. Wilkes, "Profitable services in an uncertain world," in *Proc. 2005 ACM/IEEE Conf. Supercomputing (SC'05)*, Seattle, USA, Nov. 2005.

[9] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis, "Flexible use of cloud resources through profit maximization and price discrimination," in *Proc. 2011 IEEE 27th Int'l Conf. Data Eng. (ICDE'11)*, Hannover, Germany, Apr. 2011, pp. 75–86.

[10] V. Kantere, D. Dash, G. François, S. Kyriakopoulou, and A. Ailamaki, "Optimal service pricing for a cloud cache," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 9, pp. 1345–1358, Sep. 2011.

[11] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud," in *Proc. 20th Int'l Symposium on High Performance Distributed Computing (HPDC'11)*, San Jose, USA, Jun. 2011, pp. 229–238.

[12] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. Second ACM Int'l Conf. Multimedia*, San Francisco, California, USA, Oct. 1994, pp. 15–23.

[13] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Systems*, vol. 4, no. 3, pp. 112–121, Jun. 1994.

[14] Y. Zhang, M.-Y. Wu, and W. Shu, "Adaptive channel allocation for large-scale streaming content delivery systems," *Multimedia Tools and Applications*, vol. 32, no. 3, pp. 253–273, Mar. 2007.

[15] K. C. Almeroth, "Adaptive workload-dependent scheduling for large-scale content delivery systems," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 426–439, Mar. 2001.

 ISBN: 978-1-61208-212-7 

# Design and Implementation of Cloud-based Application for Cloud Provider System with SSL Accelerator and Content Redirection

Boon Keong Seah
Advanced Computing Lab, MIMOS
Technology Park Malaysia, Malaysia
bk.seah@mimos.my

*Abstract*—**The requirement to handle large SSL connection for secure login access to applications residing in the virtual machine hosted in the cloud provider system has given rise to the need to restructure applications in the cloud utilizing the hardware load balancer or Secure Socket Layer (SSL) Accelerator. In this paper, we share our experience in designing and implementing a system in enabling SSL and non-SSL content redirection for cloud-based application. We describe the configuration and optimization rules for handling SSL transactions in the SSL Accelerator. In addition, we covered requirement by applications for sharing the HTTP session to enable seamless application access without prompting of re-login. Hence, a Single Sign-On capability in applications can be achieved.**

***Keywords-SSL; security in cloud; SSO; load balancer; SSL Accelerator.***

## I. INTRODUCTION

With increasing demands for applications to be deployed in cloud system, there is urgent need to ensure that similar security features offered in non-cloud environment be available in the cloud system as well. SSL [1][2] has been the common protocol used to provide secure communications to the web server be it in physical servers or in the virtual environment. Nevertheless, SSL also requires high CPU computation during the SSL handshake [3][6] which will impact the performance of the system hosting it. Hence, a combination of SSL and web server will consume the server resources significantly [15]. In order to distribute the SSL processing, there are two approaches. The first approach is through utilizing a load balancer and distributes the SSL process to multiple web server with SSL enabled. The load balancing of SSL can utilize several different schemes such as round robin, SSL with session, and SSL with backend forwarding [16][17]. Another approach is through utilizing the SSL Accelerator [5] as SSL reverse proxies where the SSL handshake process is offloaded [9] to the SSL Accelerator. Nevertheless, the above two approaches enforce a HTTPS connection to the client browser including accessing to the web contents.

In our implementation approach, we have designed and developed a system for enabling applications deployed in the cloud system, where only user authentication or login will require a secure channel. A session ticket will be created in the authentication system which will redirect the user to the personalized content of the applications. Each application hosted in the non-SSL channel will only be allowed access with a valid session ticket created earlier in the user authentication system. A Single Sign-On (SSO) [4] amongst applications can be realized utilizing the session ticket and hence provide a single user authentication experience.

We detailed in Section II the motivation for implementing the HTTPS and HTTP content redirection. Section III gives an overview of the system design and the operational scenario. Section IV presents the steps for the configuration and optimization rules for handling the SSL transactions in the SSL Accelerator. The implemented system is then tested and the result of the performance is shown in Section V. Section VI presents the conclusion of this paper.

## II. APPLICATION OF HTTPS AND HTTP CONTENT REDIRECTION

The system can be applied to applications hosted in the cloud where the performance and SSO of accessing the contents are of great importance. Applications such as personalized knowledge management system, media streaming, news journal content subscription and others can utilize this system to deliver contents with lower latency [15] as the contents are not accessed in the SSL protocol. In addition, the contents are protected with user authentication access rights.

In this paper, we have shown the benefits of this implementation approach in improving the performance of user access. Our implementation approach of securing the user authentication part with HTTPS whilst leaving the web content in HTTP in the cloud system is that it will enable better performance as shown in Figure 5 as compared to implementing HTTPS connection to all web contents including the user authentication as shown in Figure 4.

Major websites such as Yahoo! Mail and Facebook also have implemented this approach where only the user authentication are protected. Nevertheless the detail of such implementation approach was not published. To support high performance SSL for all contents, it involves cost and resources [15].

 ISBN: 978-1-61208-212-7

## III. SYSTEM DESIGN

### A. System Overview

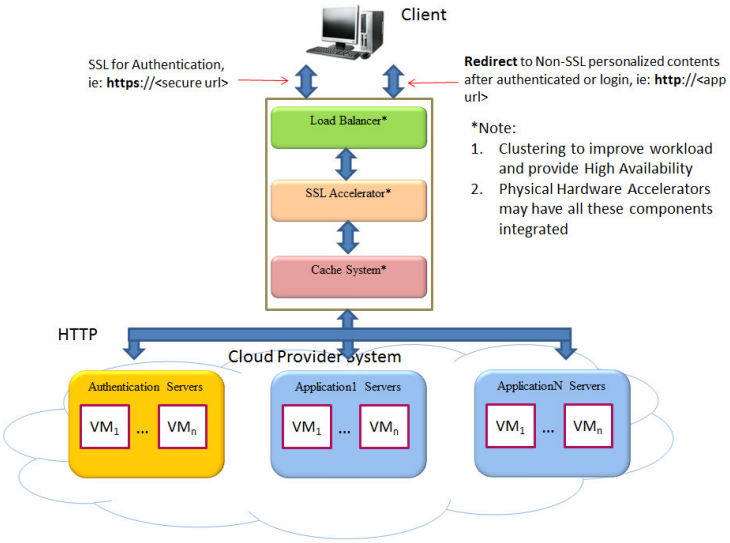


Figure 1. Overview of system design

Figure 1 shows the overview of the system design. In this design, we have the physical SSL Accelerator appliance [7][8] with integrated features such as load balancer, SSL hardware offloading and cache system acting as the **SSL reverse proxy**. The SSL Accelerator manages the HTTPS communications from clients and communicates in HTTP protocol to web servers hosted in the virtual machines.

In our design, the SSL Accelerator uses a combination of round robin load balancing and session persistent of client IP address to distribute the authentication and application Virtual Machine (VM) Servers load. The SSL Accelerator determines if the client IP address is previously connected; if it is, it returns the client to the same VM Server.

In our design, the architecture also takes into consideration the distribution of HTTPS amongst the SSL Accelerator. The SSL Accelerator [7][8] has the capability to scale with HTTPS load via workload distribution.

### B. Operational Scenario

The system, illustrated in Figure 1, works as follows:

1) HTTPS requests from client to the personalized content are intercepted by the SSL Accelerator.
2) The SSL Accelerator establishes an SSL connection to the client.
3) The SSL Accelerator checks the client's IP address is previously connected and has the session information. If the client's IP is previously connected, SSL Accelerator establishes a HTTP connection to the same VM server.
4) The SSL Accelerator checks whether the request for static contents is cached. If the content is cached, it will be serviced by the SSL Accelerator. The details of the cached configuration are given in Section IV.
5) The SSL Accelerator will assess further optimization rules configuration such as disabling HTTP TRACE and disabling accepting weak SSL cipher connection from client. The details of the optimization rules configuration are given in Section IV.
6) Upon completion of the checking by the SSL Accelerator, the HTTPS request will be forwarded as HTTP request to the pool of back-end VM applications. The back-end forwarding methods will utilize the round robin load balancing amongst the pool of VM Servers configured.
7) VM applications checks whether there is an authenticated session.
    a) If authenticated session does not exist, then it redirects to the Authentication Server for user authentication; otherwise, it permits the client access to the application without having to re-login.

## IV. IMPLEMENTATION APPROACH

The SSL Accelerator we use to implement this system is **BIG-IP LTM F5** which is a combination of load balancer, SSL hardware offloading system and cache system. In addition, the LTM F5 has in-built IDS (Intrusion Detection System) [13] to prevent network attack such as DDoS (Distributed Denial-of-Service) and IP Spoofing. The SSL Accelerator can be appliance-based, SSL Accelerator PCI card [10] based or a combinations of SSL reverse proxy applications such as POUND [11].

The authentication and application Virtual Machine (VM) Servers are deployed in MIMOS cloud. Each of the VM is allocated with 6 Virtual CPU, 16 GB of Virtual Memory, and 20 GB of Hard Disk. The VM guest OS is based on Centos 5.4 64 bit, deployed in the KVM platform [14].

As discussed earlier in Section III, in order to further improve on security and performance, the following optimization rules can be deployed in the SSL Accelerator:

1) Disable HTTP TRACE rule
    a) The HTTP TRACE request includes all HTTP headers and cookies credentials available to the web browsers. This will enable cross-site security vulnerability. In F5, the HTTP TRACE can be disabled as shown in Figure 2.

 ISBN: 978-1-61208-212-7 

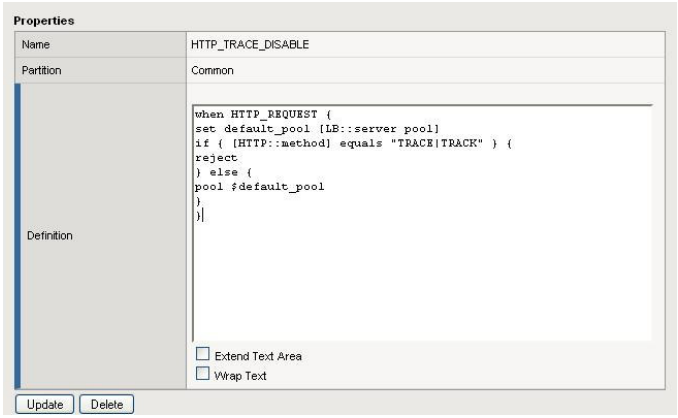


Figure 2. Disabling of HTTP TRACE

2) Disable Connection for Weak SSL Cipher

a) In the system, we also disable weak SSL cipher in the F5 for older browser as it may have SSL vulnerabilities [12]. The rule shown below will disable weak SSL Cipher connection:

- ALL:!ADH:!LOW:!EXP:!SSLv2:!NULL:HIGH:MEDIUM:RSA:RC4:

3) Cache Optimization

a) The cache optimization enables static contents caching at the SSL Accelerator which will improve the performance of the VM applications. Figure 3 illustrates the F5 configuration:

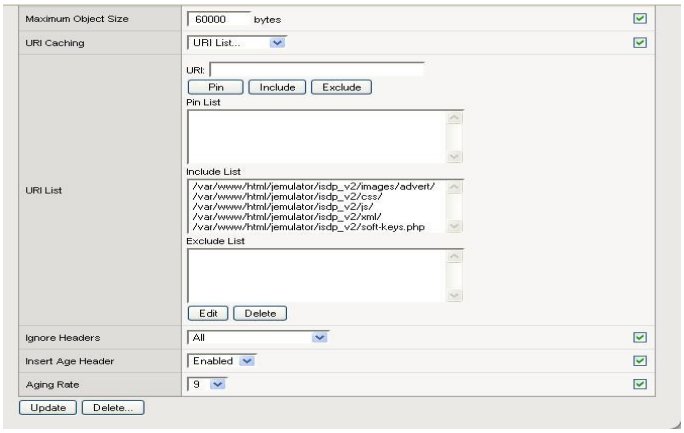


Figure 3. Enabling static content cache

## V. PERFORMANCE

In the evaluation of the performance of the system, we have deployed six web servers running applications using the Zend Framework. For the Authentication service, we also have deployed six Apache web servers serving user authentication. We prepared the system and measured the user response time for 3000 concurrent user connections with and without both the SSL Accelerator and content switching. We use JMeter as the test benchmark tool. Figures 4 and 5, shows the respective results:

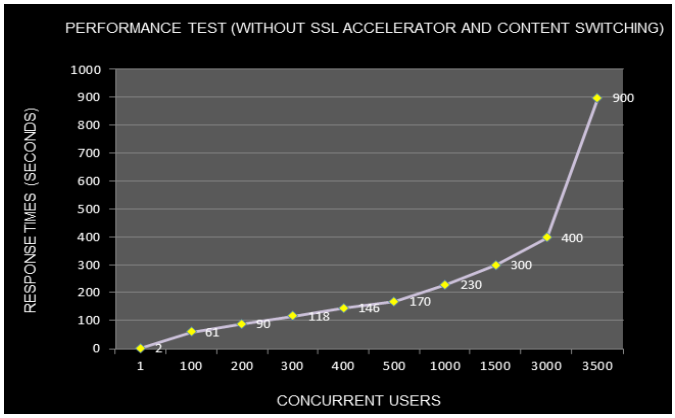


Figure 4. Concurrent user connections to the system with SSL terminating in each of the respective Apache web server.

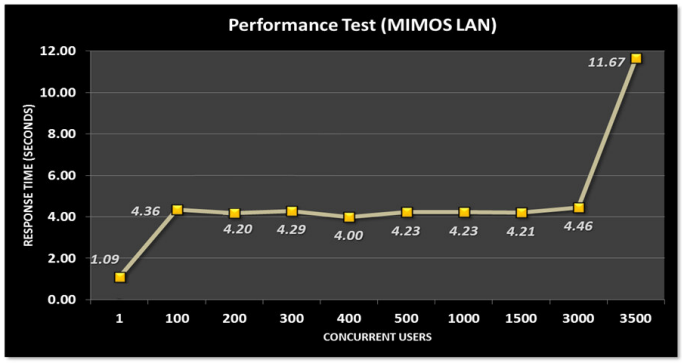


Figure 5. Concurrent user connections to the system with user authentication service using the six Apache web servers and SSL Accelerator

From Figures 4 and 5, the system we implemented shows a significant improvement of user response time. Nevertheless, in Figure 5, we noticed that the performance shows a relatively stable response time for the user connections, but degraded significantly when the concurrent user connection reaches 3000. One plausible explanation for this is that a large number of TCP connection requests are queued in the Linux network kernel buffer. This large queue will slows the network packet processing as the kernel needs to maintain the TCP connection states. In addition to the TCP connection queue size, each of the six Apache servers we used for this system implementation has a configuration of 500 maximum concurrent user connections. We did not pursue further research into

 ISBN: 978-1-61208-212-7 

modification and testing of the system implemented due to time and resource constraint.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we have successfully designed and implemented a content redirection utilizing SSL Accelerator as the SSL reverse proxy for virtual applications deployed in the cloud provider system. This approach enables the applications to offload the SSL process to the SSL Accelerator. In this system, the approach we have taken is to protect only the user authentication in HTTPS connection, while the personalized content is redirected to HTTP connection for successful authenticated user. In terms of performance, we have managed up to 3,000 concurrent SSL connections in less than 5 seconds. As part of future work, we intend to develop an SSL appliance to cater for SSL offloading; it will also be able to integrate to the cloud provider system directly for SSL subscription purposes. For user authentication protocol, federated SSO protocol such as SAML 2.0 [18] and OpenID [19] are currently being evaluated.

## REFERENCES


[1] A. O. Frier, P. Kocher, and P. C. Kaltorn, The SSL Protocol Version 3.0 draft, March 1996.

[2] J. Viega, P. Chandra, and M. Messier, Network Security with OpenSSL, 1st ed., Oreilly Publications, 2002.

[3] R. Hatsugai, T. Saito, "Load-balancing SSL Cluster Using Session Migration", 21st International Conference on Advanced Networking and Applications (AINA'07), Niagara Falls, USA, IEEE Press, Dec 2007, pp. 62-67.

[4] N. Chamberlin, Brief Overview of Single Sign-On Technology, Government Information Technology, 2000.

[5] S. Abbot, "On the Performance of SSL and an Evolution to Crytographic Coprocessors," Proc. RSA Conf., San Francisco, USA, Jan 1997.

[6] G. Apostolopoulos, V. Peris, and D. Saha, "Transport Layer Security: How much does it really cost?", IEEE INFOCOMM 1999, New York, USA, June 1999, pp. 717-725.

[7] http://www.f5.com/products/big-ip/ [retrieved: February, 2012]

[8] http://www.barracudanetworks.com/ns/products/web-site-firewall-overview.php [retrieved: February, 2012]

[9] R. Mraz, "Secure Blue: An Architecture for a Scalable, Reliable High Volume SSL Internet Server", Proc. ACSAC 2001, Australia, Dec 2001, pp. 391-398.

[10] http://www.safenet-inc.com/products/data-protection/hardware-security-modules-hsms/ [retrieved: July, 2012]

[11] http://www.apsis.ch/pound/ [retrieved: February, 2012]

[12] A. Klein, "Attacks on the RC4 stream cipher", Designs, Codes and Cryptography, vol. 48, Springer-Verlag, 2008, pp. 269-286.

[13] http://www.f5.com/pdf/white-papers/securing-enterprise-wp.pdf [retrieved: January, 2012]

[14] A. J. Younge, R. Henschel, J. T. Brown, G. Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments", IEEE CLOUD 2011, Washington, USA, July 2011, pp. 9-16.

[15] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of TLS Web servers", ACM Trans. Comput. Syst., 2006, pp. 39-69.

[16] V. M. Suresh, D. Karthikeswaran, V. M. Sudha, and D. M. Chandraseker, "Web server load balancing using SSL back-end forwarding method", IEEE ICAESM 2012, Tamil Nadu, India, March 2012, pp. 822-827.

[17] J. H. Kim, G. S. Choi, and R. D. Chita, "An SSL Back-End Forwarding Scheme in Cluster-Based Web Servers", IEEE Trans. Parallel Distrib. Syst., 2007, pp. 946-957.

[18] W. Baozhu, X. Bing, and S. Lianghong, "Design of web service single sign-on based on ticket and assertion", IEEE AIMSEC 2011, Deng Feng, China , 2011, pp. 297-300.

[19] P. Urien, "OpenID Provider based on SSL Smart Cards", Proc. CCNC 2010, Las Vegas, USA, 2010, pp. 1-2.


 ISBN: 978-1-61208-212-7 