# 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.

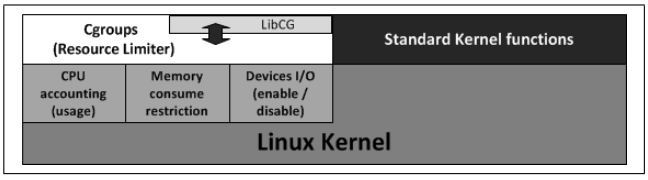


Figure 1. Overview CGROUPS in Linux kernel

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

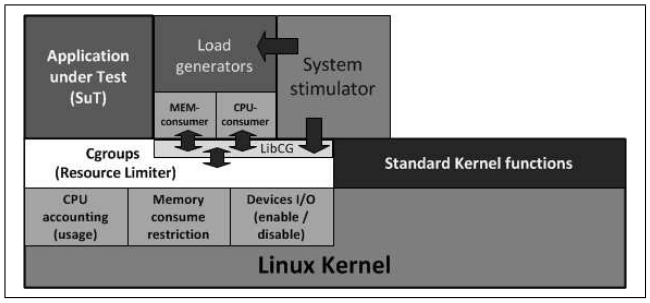


Figure 2. Sandbox regulator (CGROUPS) with load generator

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 