# Towards Fast OS Rejuvenation: An Experimental Evaluation of Fast OS Reboot Techniques

Your N. Here

Your Institution

Second Name

Second Institution

Name

Name Institution

## Abstract

Several studies have demonstrated that a fraction of software failures is caused by software aging phenomena. Software rejuvenation consists in proactively restoring a clean state of the system, via reboot/restart, to avoid the occurrence of these failures. A non-negligible fraction of OS failures is caused by software aging. Since OS rejuvenation usually requires rebooting the OS, reducing the reboot time becomes fundamental to minimize the downtime.

Different approaches have been proposed to speed up the reboot time of the OS. We refer to them as Fast OS reboot techniques. Some of these techniques are suitable candidates to implement and speed up the OS rejuvenation. Actual approaches have pros and cons to be used as a rejuvenation approach that need to be measured: the performance loss during normal operation because of extra resources required by Fast OS reboot (Performance Penalty); the extra work imposed to perform the OS rejuvenation (Rejuvenation Overhead); and the capability of the FR to restore a clean state of the system (Rejuvenation Coverage). We present the evaluation of the aforementioned issues by means of an experimental campaign to assess two Fast OS Reboot implementations for Linux OSes: kexec and phase-based reboot. Results provide useful insights to choose the most proper FR technique and suggest possible directions of improvement.

## 1 Introduction

A non-negligible fraction of software failures is caused by software aging [10, 14, 9]. Software aging is a phenomenon that leads to the accumulation of errors in the running software and operating environment by causing an increasing failure rate, degraded performance and eventually systems hang and crash. The phenomenon is due to the so called aging-related bugs [15], such as memory leak, non-terminated process/threads, file descriptors exhaustion, and memory fragmentation. Webservers [20], databases [6], network apparatus [8] and operating systems [10, 33] have manifested software aging symptoms.

Software rejuvenation [17] is the primary countermeasure to address software aging. Differently from reactive techniques, it aims to proactively prevent or to postpone aging failures. Software rejuvenation is based on stopping the system, clean up its internal state, and resume its normal operation. The growing importance of software rejuvenation to improve the system availability has been documented in recent papers [11, 1].

Nevertheless, software rejuvenation may have a non-negligible overhead on the target system [2]. In the particular case of operating systems, OS rejuvenation may impact on service disruption or downtime, since it usually requires to stop all running processes/services, reboot the OS, and thus restart/resume the processes/services. This may impact the system downtime even when fail-over procedures or redundancy are implemented [7]. For this reason, reducing the OS reboot time becomes fundamental to speed up the rejuvenation process.

The use of *skip-based* techniques is a promising approach to shorten downtime of OS reboots. Several fast OS reboot (FR) techniques have been proposed so far, and can be divided into two approaches; *reduce-based* approaches and *skip-based* ones. Reduce-based techniques make use of special hardware or special configuration to optimize the execution of one or more boot stages such as the kernel invocation [13, 19, 31, 12] and the user initialization stage [18]. On the other hand, skip-based approaches bypass several one or more boot stages [35, 25]. Skip-based approaches do not require any modification of application nor special hardware, thus being much easier to apply than the reduce-based techniques.

In this work, we comprehensively evaluate FR techniques to reveal their abilities of software rejuvenation.

We use two skip-based FR techniques: Kexec [25] and Phase-based Reboot (PBR) [35]. Although non additional overhead in using them is ideal, the techniques require extra resources and/or new software modules that may introduce some performance overhead during normal operation. To know the abilities of software rejuvenation, we identify the three following important properties of FR technique which come from a typical rejuvenation model, and conduct experiments based on the properties.

- the *Performance Penalty*: the performance loss experienced during normal operation because of the extra resources (e.g., memory, cpu and disk bandwidth) required by the FR;
- the *Rejuvenation Overhead*: the extra work imposed to the system to trigger and to carry out the rejuvenation;
- the *Rejuvenation Coverage*: the capability to restore a clean state of the system after the rejuvenation is carried out.

The results obtained reveal that FR are successful in reducing the downtime. In particular, Kexec provides an average 77% reduction when compared with the reboot; while, PBR gives an average 79% reduction if compared with VM reboot facility. These results confirm the results obtained in the previous papers [25, 35]. However, our results identify possible directions of improvement. The PBR analysis shows that, if compared with VM-Reboot, there is an increase of IO operations of 140%, on average. The percentage of the CPU usage devoted to IO also increases. As for Kexec, surprisingly no impact on the performance metrics (i.e., Throughput and Response time) is observed. On the contrary, the average system load an CPU usage using kexec is reduced by 33% and 30%, respectively.

The contributions of this paper are as follows;

- We defined metrics to quantitatively evaluate a software rejuvenation scheme.
- We evaluated two actual FR OS rejuvenation techniques based on the metrics and revealed characteristics of them in the context of software rejuvenation.
- We provided insights into design of more sophisticated FR OS techniques.

The rest of the paper is organized as follows. Section 2 provides background notions on boot stages and the related work on FR. Section 3 describes the selected FR techniques and motivates the study of these FR techniques. Section 4 and 5 presents the experimental methodology and technical details, respectively. Section 6 analyzes the results and discusses the findings. Conclusions are in Section 7.

## 2 Background and Related Work

Since the operating system (OS) plays a key role on the availability and reliability of the systems, different approaches have been proposed to develop more reliable OSes [28, 29, 16, 30]. These approaches present different mechanisms to make possible to restart faulty OS components without needing to restart the OS subsumed layers. However, these approaches are not yet in general use.

OS reboot as a proactive and reactive mechanism to deal with failures and their underlaying faults has been demonstrated effective. However, it is not uncommon to expect a downtime period during the OS reboot. Furthermore, usually the updates and patches installed on the OS (specially in the kernel) require rebooting the OS, reducing the reliability and availability of the OSes. For all these reasons, an important effort from researchers and practitioners has been dedicated to develop faster OS reboot mechanisms [35, 18, 25, 13, 19, 31, 12]. The Fast OS reboot techniques can be divided into two main categories: *skip-based* techniques which bypass one or more time consuming boot stages [35, 25]; and *reduce-based* techniques which optimize the execution of one or more boot stages such as the kernel invocation [13, 19, 31] or the user initialization stage [18]. Some techniques use specialized hardware components; e.g., in [12] exploits the DMA to reduce the kernel loading phase. In this paper, we have concentrated on the first category.

### 2.1 Skip-based Fast OS Reboot techniques

The Linux Boot process is typically divided into five main stages: i) system startup, ii) Stage 1 bootloader, iii) Stage 2 bootloader, iv) Kernel, and v) Init or User-space initialization (See Figure 1, adapted from [21]).

Briefly, the system startup phase starts in the BIOS, which checks the hardware and enumerates and initializes the local devices. The main job of Stage 1 boot loader is to find and load the second stage boot loader. The second-stage bootloader basically loads the Linux kernel, for this reason this stage is also known as kernel loader. The first and the second-stage loaders together are called Linux Loader (LILO) or Grand Unified Bootloader (GRUB), depending the Linux distribution used. Once the kernel is loaded, the kernel stage starts. The kernel stage job is to decompress the kernel and place it into the memory. Then, the kernel is booted and initialized. Once the kernel is booted and initialized, the

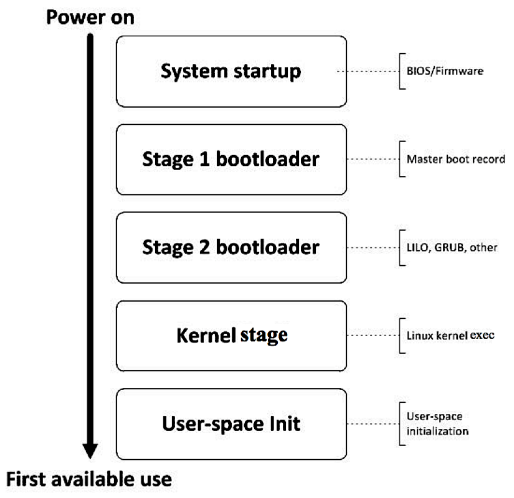

Figure 1: Linux boot stages overview

user-space application is started. After that, the Linux OS is ready to be used.

The two techniques under analysis in this paper have concentrated on skipping one or more of these time consuming stages and thus reducing significantly the boot time.

kexec [25] consists in the replacement of the kernel at runtime during normal system operation. As performing the warm reboot, the internal state of OS is refreshed but no hardware initialization is performed, namely the bios and the devices check are skipped. Live-booting can be applied in non-virtualized and in full-virtualized environments. Kexec skips the first stage of the classical Linux Boot process previously described.

In [35], the authors propose Phase-Based Reboot (PBR). PBR consists in dividing the boot in many sequences that are saved through checkpoints. To speed up the reboot, the last restorable checkpoint is restored by reproducing the same effect of the boot. PBR is based on Xen virtualization technology and in the current implementation works for para-virtualized guest OS.

Since Kexec is not possible to deploy on a virtual machine yet, it is not possible to compare them about the Performance Penalty, Rejuvenation overhead or downtime reduction of the both techniques under the same environment. For this reason, our experimental study has concentrated on measuring these three metrics for Kexec vs. classical OS reboot system call, and PBR vs. regular virtual machine (VM) restart system call from at hypervisor level.

## 2.2 Related work

In [25], kexec solution was presented and its reboot time was compared with a regular OS reboot. Kexec effectiveness was measured under different hardware architectures and different Linux distributions. The results showed a reboot time reduction up to 75%. However, other relevant metrics were not measured (e.g., performance penalty). In [35], the VM booting time reduction was also measured under a specific architecture and Linux distribution. Although, the VM boot time reduction was approx. 45%, it was also noted that the boot time reduction of PBR mainly depends on the memory size of the virtual machine.

Few papers have addressed an experimental comparison of different rejuvenation strategies [3, 27, 2]. In [3, 27], a software rejuvenation strategy for web application and Grid services was analyzed from the Performance Penalty, Rejuvenation overhead and coverage as well as downtime observed. In [2], different OS rejuvenation strategies were measured in terms of number of web requests rejected during the rejuvenation.

However, to the best of our knowledge, there is any study analyzing in detail different Fast OS rejuvenation strategies from different perspectives in order to implement a fast and effective software rejuvenation strategy for operating systems.

## 3 Internals of the Experimented FR

The FR selection has been made by considering few requirements that a FR technique should satisfy:

- *application independence*: the capability to execute the FR is not influenced by the type and the number of running applications;
- *No extra hardware needed*: avoid the integration of novel dedicated hardware;
- *Simple installation/configuration*: minimize the administrator's effort for the installation and configuration;
- *Application transparence*: avoid the reengineering of the application or of the system services.

Based on the different FR approaches, the FR techniques satisfying these criteria were Kexec and PBR. These techniques only require patching the kernel and the installation of some user-level tools.

## 3.1 Kexec

Kexec is a technique developed at Open Source Development Labs (OSDL) to allow a linux kernel loading another kernel from the main memory, by skipping the BIOS and the bootloader phases. It has been developed to support the rebooting of binary images that conform to the ELF format; hence, in principle, non-Linux kernel images could be used.

The goals of Kexec are both *(i)* to reduce the downtime of enterprise applications that need to reboot and *(ii)* to reduce the kernel developer turn-around time. In this study, we focus on the first feature of the technique.

As described in Section 2, the reboot process for Linux OS can be divided in five main stages (see Figure 1). Kexec is designed to totally skip the first two phases, i.e., the system startup, which consists of the hardware reset and the devices detection and configuration steps, and the $1^{st}$ bootloader stage, which carries out the loading of the bootable image from disk or network.

To fulfill these goals, OSDL developers have realized a kernel patch and some user-level programs. In particular, the kernel-level code can be summarized in the following system calls, which are enabled by configuring the *CONFIG_KEXEC* kernel option:

- *sys_kexec_load()* , which allows to load a kernel image and the needed kernel modules into memory;
- *sys_kexec_reboot()* , which actually boot the loaded kernel.

As for userspace code, there is a collection of tools, known as kexec-tools. The most important for our study is the *kexec*. When executed with the option *-load*, it can be used for loading the kernel into the main memory since it call the aforementioned *sys_kexec_load()* syscall. When executed with *-exec*, it executes the loaded kernel by calling the *sys_reboot()* system call with the *LINUX_REBOOT_CMD_KEXEC* command argument that prepare the later call for the *sys_kexec_reboot()* syscall.

Currently, Kexec runs on several platforms (e.g., ARM, IA-64, MIPS, PowerPC, S/390, SH, Tile, x86). However, not all the implementations are stable. For instance, with xen hypervisor kexec works well under *dom0* with old style Xenlinux patches but it does not work properly for paravirtualized OS.

## 3.2 Phase-based Reboot

The PBR concept was introduced by the research group of professor Kono from Keio University in [35]. The basic idea is simple: *(i)* separate the reboot stages, *(ii)* save the result of each stage, and *(iii)* reuse them in subsequent reboots. PBR has been designed with the aim to speed up the reboot of virtual machine and, in particular, to shorten the downtime of reboot-based recovery.

The different reboot points, called restartable candidates, are saved by means of the snapshot facilities offered by the Xen hypervisor. Then, the restartable candidate that reproduces the same effect of a regular virtual machine reboot is is restored. Of course PBR needs to guarantee that only reboot-consistent images are reused. To be reboot-consistent the restartable candidate, when restored, needs to reproduce the same effect, i.e., the same state, of a regular reboot.

To accomplish this goal PBR offers *(i)* a collection of facilities for efficiently saving different restarable candidates and *(ii)* a method to automatically check reboot-consistent images and to select the most appropiate one. These different tasks are performed by means of a collection of kernel-level patches, at guest OS, and user-level utilities that are summarized in Figure 2.

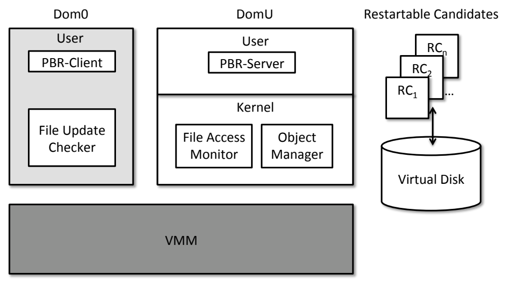

Figure 2: PBR architecture

The facilities to save restartable candidates are both at kernel-level and user-level. The guest OS kernel is explicitly modified by adding a new module: the *object manager*. This module instructs the hypervisor to skip the storing of free pages and cache objects when a snapshot is taken; it also modifies the P2M mapping table by means of the balloon driver [34]. This mechanism is useful to both reduce the amount of memory necessary to save restartable candidates and to reduce the time to restore the chosen image. In particular, the following system calls are added:

- *sys_sonicshot()* , which flushes dirty buffers and free unnecessary pages (i.e., cache and free pages);
- *sys_flush_log()* , which save the list of opened file to a given file name;
- *sys_set_operating()* , to enable or disable the monitoring activities needed by PBR.

Another important functionality added to the kernel is the *file access module*. This module monitors all files accessed during the reboot by storing the absolute path, the i-node number, and the last modication time of each file. To avoid unnecessary overhead due to the monitoring activity during operation (i.e., after a restartable candidate is taken), the file access module can be stopped by calling the *sys_set_operating()* syscall.

The userspace code is made of a simple server and some utilities. The server program waits for requests to save restartable candidates. The requests are made by a client application running at *dom0*. The server satisfies the request by calling the *sys_sonicshot()* syscall that allows to prepare the environment for taking a snapshot. The user-level utilities are a collection of scripts to save accessed files and the necessary information to select the most proper reboot-consistent image when the fast reboot is trigger.

The restoring of the most proper restartable candidate is made at dom0 by means of the *file update checker*. This modules identifies when the considered image is reboot-consistent or not by comparing the modification time of logged files and the one obtained by mounting the virtual disk.

The current implementation works for Xen 3.4.1and para-virtualized linux 2.6.18.

## 4 The Experimental Methodology

The plan and the execution of the experiments is accomplished using the Design of Experiments (DoE) methodology. DoE helps to minimize the experimental error and to get statistically significant answers in the investigation of systems or processes [23].

The first step in planning such experiments is the formulation of a clear statement of the objectives of the investigation. Then, the experimenter needs to identify the *response variables* and the *factors* of interest that can potentially affect the response variables. The value of the factors chosen in the experiments are called *levels*. When response variables, factors, and levels are identified, the test plan is completely determined by defining a list of experiments, called *treatments*

### 4.1 Objectives, Factors and Response Variables

The objectives of our study are two-fold: i) assessing if FR influences system performance during normal operation and ii) evaluating if FR has an impact on the rejuvenation overhead and rejuvenation coverage. Since the objectives are orthogonal we plan one experimental campaign to evaluate the Performance Penalty and another campaign to assess the Rejuvenation Overhead and the Rejuvenation Coverage.

The factor of interest in our analyses is the technique to rejuvenate the OS. The considered levels are: *Reboot*, *Kexec*, *VM Reboot* and *PBR*. Since the considered FR techniques work in different environments we have performed two different sets of treatments: Kexec and Reboot treatments are executed in a non-virtualized server, while VM Reboot and PBR are executed in a virtualized environment.

All treatments consists of a client application requesting web pages to a HTTP server. As for Rejuvenation Coverage treatments, we also injected memory leaks at server side in order to verify the capability to recover the wasted memory.

The response variables have been classified into server- and client-side. In particular, to evaluate the Performance Penalty we have sampled the *Throughput* and the *Response Time* at client side; while, we have sampled the *CPU*, *Free Memory*, *I/O*, and the *Load* at server-side. It is noteworthy that for PBR and VM Reboot treatments, the server-side response variables are sampled at both *guest* VM and *dom0*.

As for the response variables of the Rejuvenation Overhead and Rejuvenation Coverage campaign, we have measured the *downtime* the *recovered leaked memory*, respectively. Further details about the testbed, response variables of interest and the monitoring infrastructure are introduced and discussed in Section 5.

### 4.2 Null Hypothesis Formulation

The following null hypothesis is tested for the Performance Penalty campaign: *the FR technique does not influence the performance of the system*. The rejection of the null hypothesis means that at least one of the collected performance metrics is influenced when FR is applied.

As for the Rejuvenation Overhead a similar null hypothesis is formulated: *the FR technique does not influence the rejuvenation overhead*. In this case, if the null hypothesis is rejected it means that the downtime is influenced when FR is applied.

Finally the null hypothesis for Rejuvenation Coverage is formulated as follows: *the FR technique does not influence the rejuvenation coverage*. The rejection of the null hypothesis imply that the amount of recovered memory is influenced by the FR technique.

### 4.3 Experimental Plan and Analysis

The treatments executed for the described campaigns are divided in four plans, obtained by assigning a level to the factor of interest. The experimental plans need to adhere to the principles of DoE. Hence, we adopt the tool *JMP* (http://www.jmp.com/) to generate four plans having randomization and orthogonality properties [23] and *n* replicated treatments. The significance level $\alpha$ and the power of the test are set to 0.05 and 0.90, respectively. This means that the $Pr(reject\ H0 \mid H0\ true)$, or *type I* error, is 5% and the $1 - Pr\ (accept\ H0 \mid H0\ false)$ is 90%. The list of treatments obtained with *JMP* is reported in Table 1. Columns report the involved factor, set with a given level value. This set of treatments has been obtained considering that the minimal number of replication $n$ necessary to detect a 15% increase in the standard deviation of the response variable is $n = 16$ [23].

The Performance Penalty experimental campaign consists of 32 experiments and 32 hours of execution for each environments, i.e., virtual and non-virtual; while, the Rejuvenation Overhead and Rejuvenation Coverage campaign is made of 32 experiments and 16 hours for each considered environments. The former campaign last more than the latter since to evaluate the performance of the web server it is usually preferred to wait for the expiration of the warm up period. Indeed, during the warm-up period performance could be worst because buffers and cache are usually empty.

The method identified for the comparison of the treatments is the one-way analysis of variance (ANOVA) on each response variable. It consists in the comparison of different levels of the factor and measuring the response variable. The hypothesized model is: $y_i = \mu + F_j + \varepsilon_{i,j}$. Where $\mu$ is the overall mean, $F_j$ is the effect of the FR technique and $\varepsilon_{i,j}$ is the random experimental error.

One-way ANOVA highlights which response variable is influenced by the integration of the FR technique. However, this method only works for one response variable. When measuring more than one response variable, the chosen significance level $\alpha$ needs to be appropriately corrected by using Bonferroni's procedure [5]. This consists in dividing $\alpha$ by the number of response variables. Nevertheless, Bonferroni's correction method may become too conservative when many response variables are considered since it may need too small $\alpha$ to reject the null hypothesis. This could be the case in Performance Penalty treatments.

An alternative method that can be used with multiple response variables is the Multivariate ANOVA (MANOVA). MANOVA consists in the analysis of a linear combination of the response variables. Unfortunately, this method has some drawbacks too. Indeed, it is not particularly suited *i)* to test non linear effects and *ii)* in scenarios when there are many "weak" variables that can mask the effect of other variables.

For the aforementioned reasons, since our believe is that just few variables could be influenced by the FR, we have conducted the analysis by applying the Bonferroni's correction method. Moreover, before using ANOVA we have verified the applicability hypothesis of the test, i.e., the independence of observations, the random distribution of residuals and the homoscedasticity of variances. The first assumption is guaranteed by using the tool to randomly generate the treatments and by recreating the same experimental conditions for each treatment. Then, the other assumptions are verified to apply ANOVA.

Table 1: The experimental plans for the considered analyses

| *Exp* | *FR* | *Exp* | *FR* | *Exp* | *FR* | *Exp* | *FR* |
|---|---|---|---|---|---|---|---|
| 1 | kexec | 9 | kexec | 17 | reboot | 25 | kexec |
| 2 | reboot | 10 | reboot | 18 | reboot | 26 | reboot |
| 3 | kexec | 11 | reboot | 19 | reboot | 27 | reboot |
| 4 | kexec | 12 | kexec | 20 | kexec | 28 | kexec |
| 5 | kexec | 13 | kexec | 21 | kexec | 29 | reboot |
| 6 | reboot | 14 | reboot | 22 | reboot | 30 | kexec |
| 7 | kexec | 15 | kexec | 23 | reboot | 31 | reboot |
| 8 | kexec | 16 | reboot | 24 | kexec | 32 | reboot |

**(a) Performance Penalty campaign in non-virtual environment**

| *Exp* | *FR* | *Exp* | *FR* | *Exp* | *FR* | *Exp* | *FR* |
|---|---|---|---|---|---|---|---|
| 1 | pbr | 9 | pbr | 17 | pbr | 25 | pbr |
| 2 | vmr | 10 | vmr | 18 | vmr | 26 | vmr |
| 3 | vmr | 11 | vmr | 19 | vmr | 27 | vmr |
| 4 | vmr | 12 | pbr | 20 | pbr | 28 | pbr |
| 5 | pbr | 13 | pbr | 21 | pbr | 29 | vmr |
| 6 | pbr | 14 | pbr | 22 | vmr | 30 | pbr |
| 7 | pbr | 15 | pbr | 23 | vmr | 31 | vmr |
| 8 | pbr | 16 | vmr | 24 | pbr | 32 | vmr |

**(b) Performance Penalty campaign in virtual environment**

## 5 Technical Details

### 5.1 The Experimental Testbed

jimgray: PowerEdge T310 8GB RAM Intel Xeon X3430 2.4 GHz, 8M Cache Turbo 1TB 7.2K RPM SATA
The virtual environment testbed is made up of two machines: (i) a server, equipped with an Intel Pentium 4 3.2 GHz CPU with Hyper-Threading, 4 GB RAM, 1 Gb/s Network Interface and (ii) a client, equipped with an Intel Pentium 4 2.4 GHz CPU, 2 GB of RAM, 1 Gb/s Network Interface. The Guest OS is the paravirtualized Linux version 2.6.18 provided by Xen and equipped with PBR modules; while, the dom0 run the Xen 3.4.1 hypervisor.

The non-virtual environment testbed is made of a (i) server with XX GB of RAM and X CPU, 1 Gb/s Network Interface and running Scientific Linux 2.6.32 EL and a (ii) client with 32 GB of RAM and Intel Xeon E5620 (16 core) 2.40GHz, 1 Gb/s Network Interface, also and running Scientific Linux 2.6.32.

On the server machines of both virtual and non-virtual environments we run an Apache HTTP server 2.4.2; while at the client-side we use $httperf$ [24] as traffic generator to request web pages of different size, i.e., few bytes to some MB.

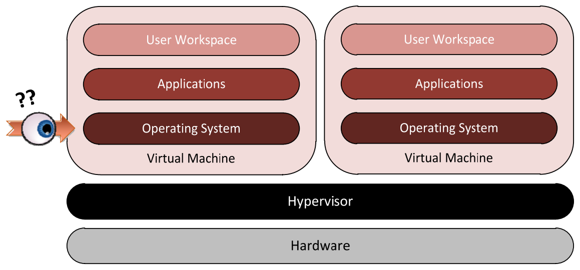

Figure 3: The single OS view is not appropriate in virtual environments. Figure taken from [26]

## 5.2 Response Variables Description

The Performance Penalty have been evaluated at both client- and server-side in terms of the: *Throughput* and *Response Time*, and , *CPU I/O*,*Free Memory* and *Load* (see Table 5.2). These metrics are collected every $\delta$ seconds.

The Throughput is measured at the application-level and as the average number or reply over the number of requests in the interval of time. This may be influenced by the extra software required by the FR technique running at server-side. As for the response time, it is measured as the total time elapsed between the first byte sent by the client and the first byte received by the server.

At server-side the CPU, the Load, the Free Memory and the IO are collected. The CPU accounts for the instantaneous consumption of the CPU for all running processes; the Load, instead is the exponential moving average of the number of processes ready to be executed, giving more weight to the last 15 minutes samples; the Free Memory is the amount of all available RAM including buffer and caches; the IO accounts for the number of blocks read from and write to the disk since last reboot. In the case of virtual environments, the VMM-level also needs to be monitored to capture the Performance Penalty since extra resources not consumed by the guest OS can be consumed by the VMM (see Figure 3).

As for the Rejuvenation Overhead and the Rejuvenation Coverage, we monitored the *Downtime* and the *Recovered Memory*, respectively. The downtime is evaluated by measuring the seconds needed by the FR technique to reboot the system. This also includes the time to prepare to reboot but not the time to set up the user environment, namely to run all the user-level processes. To compute the recovered memory, we injected some memory leak at kernel level, then we evaluate if the amount of wasted memory has been recovered after the reboot by simply comparing the free memory values before and after the treatment.

Table 2: Response variables to evaluate Performance Penalty, Rejuvenation Overhead and Rejuvenation Coverage

| Metrics | Unit of measure |
|---|---|
| Throughput (T) | # Response per sec |
| Response time (RT) | Seconds |

**(a) Client-side response variables**

| Metrics | Unit of measure |
|---|---|
| CPU | Percentage |
| Load | # processes in ready queue |
| Free Memory (FM) | Kilobyte |
| IO | #Block from/to disks |
| Downtime (D) | seconds |
| Recovered Memory (RM) | percentage |

**(b) Server-side response variables**

It is noteworthy that the response variables selected to conduct the analysis are both coarse-grained and fine-grained. The former, i.e., throughput and response time and downtime, have the primary purpose to evaluate the overall impact of each FR technique at the application level. Then, to find out why a particular FR technique has the observed behavior, the fine-grained response variables, such as Free Memory, IO and CPU and load, can be used to conduct a deeper analysis trying to isolate the root cause.

## 5.3 Monitoring infrastructure

To quantitatively capture the aforementioned response variables we instrumented the target systems at VMM- OS- and application-level.

We use existing OS and VMM monitoring tools to measure the metrics at the OS- and VMM-level, such as *top*, *free* and *vmstat*. Instead, we exploit httperf built-in facilities to measure at client-side the Throughput and the Response Time.

As for the rejuvenation coverage campaign, the injection of kernel-level memory leaks has been performed by a simple kernel module. The injection events consists of the allocation and the initialization of a fixed amount of memory and the times of the injection events are Poisson process. The treatments have been executed with the tool configured to inject 1KB of memory leak having the inter-arrival time of the injection events exponentially distributed with $\lambda = 1\ minute$.

The side effects of the monitoring infrastructure has also been evaluated. The impact on the application is negligible since these tools just read some small files at a predefined interval of $\delta = 15$ seconds.

## 6 Results

In this section, we present the results obtained during the experiments with Kexec and PBR approaches. The experiments have been focused on the two metrics of interest: Performance Penalty and Rejuvenation overhead. Due to the incompatible scenarios of Kexec and PBR, we cannot compare each other. The analysis has been focused on measuring the Performance Penalty and Rejuvenation overhead of the Fast OS reboot techniques under study with the same scenario and OS without the techniques deployed.

### 6.1 Kexec

#### 6.1.1 Performance Penalty

AGGIORNARE
In order to measure the Performance Penalty introduced by Kexec, we collect different performance and system metrics during normal operation with Kexec installed on the OS and without it.

The collected data has been divided into two types or perspectives: user-based metrics and system-based metrics. User-based metrics are composed of Throughput and Response time. On the other hand, system-based metrics are composed of System load, CPU usage, I/O usage, and Free Memory.

In Tables **??** and **??**, The averaged user-based metrics and system-based metrics are presented, respectively.

In Table **??**, we observe that the throughput difference is negligible. However, It seems that the difference in terms of Response time can be significant. In the case of Server-based metrics (see Table **??**, we observe similar results. But, it is interesting to observe that the Load and I/O of the kexec OS system are lower than the classical OS system.

In order to confirm or reject that intuition, one-way ANOVA test is used to test if the difference on the average throughput and response time of Kexec OS and classical OS are significant at $\alpha = 0.05$. However, it is required to use the Bonferroni correction [22] since we are conducting multiple independent statistical tests simultaneously. Even when the $\alpha$ value is appropriate for each individual comparison, it is not for the set of all comparisons. In order to avoid false positives, the $\alpha$ value needs to be lowered appropriately, taking into account the number of comparisons (i.e., $n = 6$). So, based on the Bonferroni's correction, $\alpha = 0.05/6 = 0.0083$. Table 3 presents the one-way ANOVA results of user-based metrics. The

Table 3: Results of the Performance Penalty analysis

| **Source** | **df** | **SS** | **MS** | **F** | **P-value** |
|---|---|---|---|---|---|
| rej. tech. | 1 | 1.91E-01 | 6.92E-01 | 4.17 | |
| error | 30 | | | | |
| total | 30 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(a) One-way ANOVA for Load**

| **Source** | **df** | **SS** | **MS** | **F** | **P-value** |
|---|---|---|---|---|---|
| rej. tech. | 1 | 1.67E01 | 4E-04 | 4.1708 | |
| error | 30 | xx | | | |
| total | 30 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(b) One-way ANOVA for CPU**

| **Source** | **df** | **SS** | **MS** | **F** | **P-value** |
|---|---|---|---|---|---|
| rej. tech. | 1 | 1.67E01 | 4E-04 | 4.1708 | |
| error | 30 | xx | | | |
| total | 30 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(c) One-way ANOVA for IO**

| **Source** | **df** | **SS** | **MS** | **F** | **P-value** |
|---|---|---|---|---|---|
| rej. tech. | 1 | 1.67E01 | 4E-04 | 4.1708 | |
| error | 30 | xx | | | |
| total | 30 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(d) One-way ANOVA for Memory**

| **Source** | **df** | **SS** | **MS** | **F** | **P-value** |
|---|---|---|---|---|---|
| rej. tech. | 1 | xxx | xxx | xxx | |
| error | 30 | xx | | | |
| total | 30 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(e) One-way ANOVA for Throughput**

| **Source** | **df** | **SS** | **MS** | **F** | **P-value** |
|---|---|---|---|---|---|
| rej. tech. | 1 | 1.67E01 | 4E-04 | 4.1708 | |
| error | 30 | xx | | | |
| total | 30 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(f) One-way ANOVA for Response Time**

Table 4: Welch's test at Server-based metrics

| **Load3** | | | | |
|---|---|---|---|---|
| F-ratio | Test t | Num. df | T crit. | P-value |
| 3.801 | 7.8938 | 22 | 2.0738 | $2.3628*10^{-8}$ |
| **CPU** | | | | |
| F-ratio | Test t | Num. df | T crit. | P-value |
| 17.7381 | 4.965 | 17 | 2.1098 | 0.000117 |
| **I/O** | | | | |
| F-ratio | Test t | Num. df | T crit. | P-value |
| 63.125 | 3.1508 | 15 | 2.1314 | 0.00659 |
| **Free Mem.** | | | | |
| F-ratio | Test t | Num. df | T crit. | P-value |
| 82.338 | 0.0467 | 15 | 2.1314 | 0.9633 |

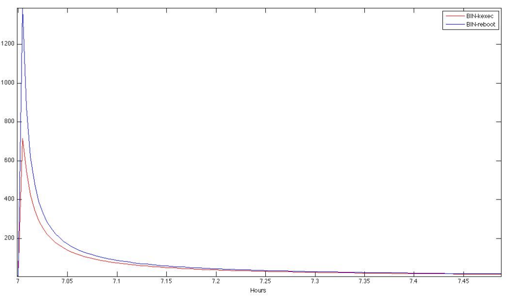

Figure 4: Example of I/O peak during warm-up period

ANOVA results for user-based metrics indicated null hypothesis cannot be rejected ($p-value \geq 0.0083$) for both response variables of interest. This result becomes relevant because indicates that the user-perceived performance of the system is not affected by installing Kexec.

In the case of system-based metrics, the hypothesis of homoscedasticity required to perform one-way ANOVA is not satisfied. For this reason, we have computed the Welch's test which does not requires homoscedasticity hypothesis be satisfied. Table 4 presents the results of Welch's test. We observe significant differences according to the statistic test results for *Load3*, *CPU*, and *I/O*. So, we can reject the null hypothesis, indicating that these system metrics are significantly affected by the factor under study (i.e., with/without Kexec). On the other side, memory free is not significantly different between both levels.

In order to explain the significant difference of *Load3*, *CPU*, and *I/O*, we have studied the samples collected along the experimental runs (see I/O example of one of the runs in Figure 4. During the first minutes of the each run (warmup period), a peak was observed for all these three metrics. Hence, the greater values at the beginning can bias the average values for the experiments. In any case, these differences are not enough to affect the performance of the system from the user perspective, which is the desirable global result.

Table 5: Results of the Rejuvenation Overhead and Rejuvenation Coverage analysis

| **F ratio** | **Test t** | **df** | **Den. df** | **P-value** |
|---|---|---|---|---|
| 8.46E03 | 9.20E01 | 1 | 24.83 | 1.00E-04 |

**(a) Welch's Test for Downtime**

| **Source** | **df** | **SS** | **MS** | **F** | **P-value** | |
|---|---|---|---|---|---|---|
| rej. tech. | 1 | xxx | xxx | 2.00E-01 | 4.17E00 | 4E-04 |
| error | 30 | xx | | | | |
| total | 31 | xx | | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | $95\%CI^{-}$ | $95\%CI^{+}$ |
|---|---|---|---|---|
| kexec | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| reboot | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(b) One-way ANOVA for Recovered Memory TO UPDATE**

### 6.1.2 Rejuvenation Overhead and Coverage

The second campaign has been executed to investigate the Rejuvenation Overhead and Rejuvenation Coverage. We have conducted two statistical tests to verify if there is any statistical evidence showing that the factor FR technique has a statistical influence on the downtime and on the recovered memory.

As for the downtime analysis, the hypothesis of homoscedasticity required to perform ANOVA is not satisfied. In fact, three tests (i.e., O-Brien, Brown-Forsythe and Levene's tests) out of five that are usually performed to verify that the variances of groups are homogeneous have rejected the null hypothesis *variances into groups are equal*. However, the tests not rejecting the null hypothesis have a really p-value quite close to the critical value (i.e., 0.05). For these reasons we have applied the Welch's test that does not assume homoscedasticity.

The results of the statistical tests are summarized in Table 5. The upper table shows that when the rejuvenation is accomplished using Kexec the downtime is in average 25.4 seconds; while, the downtime due to the reboot is in average 117.5 seconds. Hence, there is about a 77% reduction of the downtime when applying Kexec.

As for the Rejuvenation coverage, the wasted memory due to the injected memory leaks has been totally recovered in all the treatments. Indeed, the FR has no impact on the capability to recover from memory leak at kernel level.

## 6.2 Phased-based reboot

### 6.2.1 Performance Penalty

In this section, we present the results of the user and system metrics used to measure the performance penalty of the Phase-based reboot technique (PBR). Tables 6 and 7

Table 6: User-based metrics

| | PBR OS | Regular VM OS | Diff. |
|---|---|---|---|
| Avg. Throughput (req/sec) | 32.525 | 32.45 | 0.075 |
| Std. Dev. Throughput | 0.1437 | 0.121 | |
| Avg. Response time (ms) | 861.63 | 872.81 | 11.1875 |
| Std. Dev. Response Time | 10.763 | 9.893 | |

Table 7: Server-based metrics

| | PBR OS | Regular VM OS | Diff. |
|---|---|---|---|
| DOMU Avg. Load3 | 0.023 | 0.005 | 0.017 |
| DOMU Std. Dev. Load3 | 0.00544 | 0.0018 | |
| DOMU Avg. CPU | 2.072 | 1.6275 | 0.4453 |
| DOMU Std. Dev. CPU | 0.101 | 0.000148 | |
| DOMU Avg. I/O | 92.28 | 67.11 | 25.16 |
| DOMU Std. Dev. I/O | 3.029 | 52.316 | |
| DOMU Avg. Free Memory | 1701909.57 | 1682972.15 | 18937.41 |
| DOMU Std. Dev. Free Memory | 1786.15 | 46979.73 | |
| DOM0 Avg. Load3 | 0.0033 | 0.0015 | |
| DOM0 Std. Dev. Load3 | 0.0032 | 0.003 | |
| DOM0 Avg. CPU | 0.3548 | 0.3375 | 0.017 |
| DOM0 Std. Dev. CPU | 0.14 | 0.05 | |
| DOM0 Avg. I/O | 43.23 | 43.08 | 0.14 |
| DOM0 Std. Dev. I/O | 9.15 | 1.499 | |
| DOM0 Avg. Free Memory | 181021.088 | 124050.661 | 56970.427 |
| DOM0 Std. Dev. Free Memory | 32014.66 | 3680.35 | |

summarize the average results obtained for user and system (Dom U and Dom 0) metrics, respectively.

In the case of User-based metrics, the hypothesis of homoscedasticity is satisfied and one-way ANOVA is a valid test to probe the null hypothesis:*the user metrics have the same mean*. Table 8 presents the ANOVA results for throughput and response time. We observe how the null hypothesis cannot be rejected in the first case, but it can in the second case. So, it can be concluded that the average response time is affected by the integration of PBR solution on the virtualized environment.

In the case of system metrics, we have observed that the Dom U *Load3* results satisfy the hypothesis of homoscedasticity, but the rest of metrics do not. Table 9 presents the one-way ANOVA test of Load3 at Dom U. We observe that the null hypotheis cannot be rejected with at $\alpha = 0.05$.

Table 8: One-way ANOVA at User-based metrics

**Throughput**

| Source | df | F | P-value | F crit |
|---|---|---|---|---|
| Between | 1 | 2.547 | 0.1209 | 4.1708 |
| Within | 30 | | | |

**Response Time**

| Source | df | F | P-value | F crit |
|---|---|---|---|---|
| Between | 1 | 9.369 | 0.0046 | 4.1708 |
| Within | 30 | | | |

Table 9: One-way ANOVA at System-based metric

**Load3**

| Source | df | F | P-value | F crit |
|---|---|---|---|---|
| Between | 1 | 2.376 | 0.13362 | 4.1708 |
| Within | 30 | | | |

Table 10: Welch's test of Server-based metrics at Dom U/Dom 0

| F-ratio | Test t | Num. df | T crit. | P-value |
|---|---|---|---|---|
| **DOMU-Load3** | | | | |
| 8.888 | 12.323 | 18 | 2.1009 | $3.284*10^{-10}$ |
| **DOMU-CPU** | | | | |
| 9.144 | 3.906 | 18 | 2.1009 | 0.00103 |
| **DOM0-CPU** | | | | |
| 8.5711 | 27 | 15 | 2.13144 | $3.9305*10^{-14}$ |
| **DOMU-I/O** | | | | |
| 17.2705 | 13.6089 | 16 | 2.1199 | $3.2575*10^{-10}$ |
| **DOM0-I/O** | | | | |
| 37.277 | 0.06375 | 16 | 2.1199 | 0.9499 |
| **DOMU-Free Mem.** | | | | |
| 691.803 | 1.611 | 15 | 2.13144 | 0.1279 |
| **DOM0-Free Mem.** | | | | |
| 75.669 | 7.0683 | 15 | 2.13144 | $3.8195*10^{-6}$ |

Table 10 presents the Welch's test results for the rest of metrics at Dom U and Dom 0. At Dom U, we can reject the null hypothesis in all the cases. The reason is similar that presented in the Kexec case. At Dom 0, we observe an interesting behavior. The I/O metric results indicate that there is no significant difference on using PBR or not. However, in the rest of metrics the null hypothesis can be rejected. So, it is concluded from the analysis conducted that PBR affects the system metrics, like kexec does. However, PBR has a negative impact on the user metric Response time. This limits the effectiveness of the solution. On the other side, as it has been indicated, it is not possible to deploy kexec in a VM. So, PBR becomes an effective substitute, at least until kexec would be available for these relevant scenarios.

### 6.2.2 Rejuvenation Overhead and Coverage

As for the downtime analysis in the virtual environment, using PBR instead of the VM reboot provides an average of 79% downtime reduction. Indeed, the average downtime using PBR and VM reboot is 5.03 seconds and 23.9

Table 11: PBR results for Rejuvenation Overhead and Rejuvenation Coverage analysis

| Source | df | SS | MS | F | P-value |
|---|---|---|---|---|---|
| rej. tech. | 1 | xxx | xxx | 2.00E-01 | 4.17E00 |
| error | 30 | xx | | | |
| total | 31 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(a) One-way ANOVA for Downtime**

| Source | df | SS | MS | F | P-value |
|---|---|---|---|---|---|
| rej. tech. | 1 | xxx | xxx | 1.67E01 | 4.00E-04 |
| error | 30 | xx | | | |
| total | 31 | xx | | | |

*Means for one-way ANOVA*

| Level | Avg | Std Dev | 95%$CI^-$ | 95%$CI^+$ |
|---|---|---|---|---|
| pbr | 1.97E-01 | 2.93E-03 | 1.91E-01 | 2.03E-01 |
| vmr | 2.14E-01 | 3.04E-03 | 2.08E-01 | 2.22E-01 |

**(b) One-way ANOVA for Recovered Memory**

seconds, respectively. The hypothesis of homoscedasticity required to perform ANOVA is satisfied, all the tests cannot rejected the null hypothesis *variances into groups are equal*. Hence, the results of the statistical analysis by applying ANOVA, which are summarized in Table 11, highlight that the FR technique influences the downtime.

As for the Rejuvenation Coverage, the wasted memory due to the injected memory leaks cannot be totally recovered using the PBR. Indeed, if the snapshot is taken when an injection has been performed, this wasted memory is saved in the image. Hence, when the image is restored when the memory leak cannot be cleaned. This behavior is highlighted by the analysis summarized in Table 11. There is a statistical evidence ($p-value \ll \alpha$) that PBR cannot recover all the injected memory leaks.

## 7 Conclusion

## Acknowledgments

## References

[1] ALONSO, J., BOVENZI, A., LI, J., WANG, Y., RUSSO, S., AND TRIVEDI, K. Software Rejuvenation - Do IT & Telco Industries Use It? In *The 4th International Workshop on Software Aging and Rejuvenation* (2012).

[2] ALONSO, J., MATIAS, R., VICENTE, E., MARIA, A., AND TRIVEDI, K. A comparative experimental study of software rejuvenation overhead. *Performance Evaluation*, 0 (2012), –.

[3] ALONSO, J., SILVA, L., ANDRZEJAK, A., SILVA, P., AND TORRES, J. High-available grid services through the use of virtualized clustering. In *GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 34–41.

[4] ALONSO, J., TORRES, J., BERRAL, J. L., AND GAVALDA, R. Adaptive on-line software aging prediction based on machine learning. *Dependable Systems and Networks, International Conference on 0* (2010), 507–516.

[5] BLAND, J. M., AND ALTMAN, D. G. Multiple significance tests: the bonferroni method. *BMJ 310*, 6973 (January 1995), 170.

[6] BOVENZI, A., COTRONEO, D., PIETRANTUONO, R., AND RUSSO, S. On the aging effects due to concurrency bugs: A case study on mysql. In *Accepted for publication in Procs. IEEE Int'l. Symp. Software Reliability Engineering* (2012).

[7] CANDEA, G., KAWAMOTO, S., FUJIKI, Y., FRIEDMAN, G., AND FOX, A. Microreboot - a technique for cheap recovery. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), USENIX Association, pp. 3–3.

[8] CISCO SYSTEMS, I. Cisco security advisory: Cisco catalyst memory leak vulnerability. document id: 13618, 2001.

[9] COTRONEO, D., GROTTKE, M., NATELLA, R., PIETRANTUONO, R., AND TRIVEDI, K. S. An empirical investigation of fault types in space mission system software. In *Submitted to review* (2013).

[10] COTRONEO, D., NATELLA, R., PIETRANTUONO, R., AND RUSSO, S. Software aging analysis of the linux operating system. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on* (nov. 2010), pp. 71 –80.

[11] COTRONEO, D., NATELLA, R., PIETRANTUONO, R., AND RUSSO, S. Software aging and rejuvenation: Where we are and where we are going. In *Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on* (29 2011-dec. 2 2011), pp. 1 –6.

[12] EMBEDDED LINUX. http://elinux.org/DMA_Copy_Of_Kernel_On_Startup.

[13] EMBEDDED LINUX. http://elinux.org/Reordering_of_driver_initialization.

[14] GROTTKE, M., NIKORA, A., AND TRIVEDI, K. An empirical investigation of fault types in space mission system software. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on* (28 2010-july 1 2010), pp. 447 –456.

[15] GROTTKE, M., AND TRIVEDI, K. S. Fighting bugs: Remove, retry, replicate, and rejuvenate. *Computer 40* (2007), 107–109.

[16] HERDER, J. N., BOS, H., GRAS, B., HOMBURG, P., AND TANENBAUM, A. S. Failure resilience for device drivers. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (Washington, DC, USA, 2007), DSN '07, IEEE Computer Society, pp. 41–50.

[17] HUANG, Y., KINTALA, C., KOLETTIS, N., AND FULTON, N. D. Software rejuvenation: Analysis, module and applications. *Fault-Tolerant Computing, International Symposium on 0* (1995), 0381.

[18] HUNT, J. Parallelize linux system services to improve boot speed.

[19] JOO, Y., RYU, J., PARK, S., AND SHIN, K. G. FAST: quick application launch on solid-state drives. In *Proceedings of the 9th USENIX conference on File and stroage technologies* (Berkeley, CA, USA, 2011), FAST'11, USENIX Association, pp. 19–19.

[20] LI, L., VAIDYANATHAN, K., AND TRIVEDI, K. S. An approach for estimation of software aging in a web server. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 91–100.

[21] M. TIM JONES. *Inside the Linux boot process http://www.ibm.com/developerworks/linux/library/l-linuxboot/.*

[22] MILLER, R. G. *Simultaneous Statistical Inference*. Springer-Verlag, New York, 1991.

[23] MONTGOMERY, D. *Design and Analysis of Experiments*. Student solutions manual. Wiley, 2008.

[24] MOSBERGER, D., AND JIN, T. httperf–a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev. 26*, 3 (Dec. 1998), 31–37.

[25] PFIFFER, A. Reducing System Reboot Time With kexec. http://www.osdl.org/docs/ reducing_system_reboot_time_with_kexec.pdf, 2003.

[26] SHIELDS, G. New best practices in virtual and cloud management. www.vmware.com/files/pdf/vmware-best-practices-cloud-management-performance-capacity-compliance-workload-automation.pdf.

[27] SILVA, L., ALONSO, J., AND TORRES, J. Using virtualization to improve software rejuvenation. *Computers, IEEE Transactions on 58*, 11 (nov. 2009), 1525 –1538.

[28] SUNDARARAMAN, S., SUBRAMANIAN, S., RAJIMWALE, A., ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H., AND SWIFT, M. M. Membrane: Operating System Support for Restartable File Systems. In *Proceedings of the 8th Conference on File and Storage Technologies (FAST '10)* (San Jose, California, February 2010), pp. 281–294.

[29] SWIFT, M. M., ANNAMALAI, M., BERSHAD, B. N., AND LEVY, H. M. Recovering device drivers. *ACM Trans. Comput. Syst. 24* (November 2006), 333–360.

[30] SWIFT, M. M., MARTIN, S., LEVY, H. M., AND EGGERS, S. J. Nooks: an architecture for reliable device drivers. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop* (New York, NY, USA, 2002), EW 10, ACM, pp. 102–107.

[31] UCL. http://www.oberhumer.com/opensource/ucl/.

[32] VAIDYANATHAN, K., AND TRIVEDI, K. A comprehensive model for software rejuvenation. *IEEE Trans. Dependable Secur. Comput. 2*, 2 (2005), 124–137.

[33] VAIDYANATHAN, K., AND TRIVEDI, K. S. A measurement-based model for estimation of resource exhaustion in operational software systems. In *Proceedings of the 10th International Symposium on Software Reliability Engineering* (Washington, DC, USA, 1999), ISSRE '99, IEEE Computer Society, pp. 84–93.

[34] WALDSPURGER, C. A. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev. 36*, SI (Dec. 2002), 181–194.

[35] YAMAKITA, K., YAMADA, H., AND KONO, K. Phase-based reboot: Reusing operating system execution phases for cheap reboot-based recovery. In *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on* (june 2011), pp. 169 –180.