A Cooperative Approach to Virtual Machine Based Fault Injection

Thomas Naughton[1], Christian Engelmann[1], Geoffroy Vallée[1], Ferrol Aderholdt[1], and Stephen L. Scott[1,2]

[1] Oak Ridge National Laboratory*
Computer Science and Mathematics Division
Oak Ridge, TN 37831, USA
[2] Tennessee Tech University
Computer Science
Cookville, TN, 38505, USA

Abstract. Resilience investigations often employ fault injection (FI) tools to study the effects of simulated errors on a target system. It is important to keep the target system under test (SUT) isolated from the controlling environment in order to maintain control of the experiement. Virtual machines (VMs) have been used to aid these investigations due to the strong isolation properties of system-level virtualization. A key challenge in fault injection tools is to gain proper insight and context about the SUT. In VM-based FI tools, this challenge of target context is increased due to the separation between host and guest (VM). We discuss an approach to VM-based FI that leverages virtual machine introspection (VMI) methods to gain insight into the target's context running within the VM. The key to this environment is the ability to provide basic information to the FI system that can be used to create a map of the target environment. We describe a proof-of-concept implementation and a demonstration of its use to introduce simulated soft errors into an iterative solver benchmark running in user-space of a guest VM.

Keywords: fault injection, virtualization, virtual machine introspection, resilience tools

1 Introduction

Tools for controllably experimenting with synthetic failures are an essential element of resilience investigation. These tools generally employ some form of software implemented fault injection (SWIFI) since it is highly adaptable, in contrast to hardware

* Corresponding author: T. Naughton naughtont@ornl.gov. This manuscript has been authored by UT-Battelle, LLC under Contract No.DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

based approaches [11]. However, low-level hardware approaches have some advantages for performing tests that can originate at the lowest layers of the system. System-level virtualization has been explored as a way to combine the advantages of SWIFI with the low-level hardware oriented approaches using virtual machines (VMs) [14, 18, 21].

There are several advantages to using virtual machines with fault injection. The use of virtualization allows for strong isolation between the system under test (SUT) and control environment. The VMs provide a basis to customize the target environment and setup repeatable testing configurations. The strong isolation provided by the VMs can be beneficial for resilience experiments that might include tests that compromise the overall investigation environment, e.g., data corruption, high crash rates.

A major challenge of virtual machine based fault injection (VMFI) is providing adequate context about the target to inform site selection choices. Additionally, the target's context must be sufficiently understood in order to monitor the target's status and interpret the effects of injected errors. The lack of insight into the target (guest) context is a common issue with virtualization and emerges in many instances where information maintained within the guest's context would be useful outside the guest VM, e.g., process monitoring. The technique of virtual machine introspection (VMI) was developed to overcome just these types of challenges and has been applied to performance monitoring and security.

We have used VMI methods with VM-based fault injection to bridge the gap between the target (in guest) and controller (outside guest). We describe the approach and demonstrate a proof-of-concept experiment where we can perform fault injection on a process in a VM using commands from the host (outside the VM). This approach maintains the strong isolation of VMFI and leverages VMI methods to gain target context. The primary contributions of this paper are:

- The presentation of tools for HPC Resilience investigations that support experiments at both user and kernel levels, which can be performed with strong separation between control and system under test environments;
- A description of a cooperative VM-based fault-injection (FI) mechanism, which includes a discussion of how VMI can benefit FI;
- The demonstration of proposed FI mechanism to study soft error resilience in iterative solver benchmark running in user-space of guest VM.

2 Background

2.1 Virtualization

The virtualization of physical hardware enables a privileged software layer to multiplex the underlying physical resources. This management layer is called a virtual machine monitor (VMM), or *hypervisor*, and is responsible for providing VMs with efficient, controlled access to the physical resources [23, 20]. The VMM runs on a *host* machine, and a VM runs on the VMM. The VM is often termed the *guest* and the operating system (OS) running in the VM is termed the *guest operating system* (or guest OS). There are two categories of VMMs that are distinguished by their position in the software stack with respect to the physical hardware [20]: i) executes directly on the hardware (*type-I*), ii) executes atop or within a host OS (*type-II*).

There are several open-source and commercial offerings for virtualization. Palacios [13] is a VMM that has been developed specifically for use in high-performance computing (HPC) environments. It can be embedded within the Kitten light-weight kernel or Linux OS. The implementation uses hardware extensions available in modern x86 processors to provide efficient virtualization. Palacios runs on standard x86 commodity clusters and Cray XT/XK supercomputers. Palacios is currently being used as part of the Hobbes OS research project [2].

2.2 Virtual Machine Introspection

Virtual machine introspection allows for a guest's internal state to be exposed to an external viewer, commonly another VM [19], the VMM [1], or a process on the host [8]. Because the VM is executing on a software or hardware abstraction of physical resources, the amount of state exposed by VMI is extensive, ranging from device registers to the memory of the guest. This allows for the external software to both observe and modify the guest's state. However, the view of the guest's state is often difficult to understand because of the "semantic gap" [4]. To overcome this obstacle, researchers often create a bridge across the semantic gap by means of a memory map of a particular process or the guest OS. An example of this bridge with respect to Linux is the `System.map` file, which holds a significant amount of information including the virtual address of the various functions, data structures, and other data residing within the kernel.

2.3 Fault Injection

Virtualization offers several useful mechanisms for implementing fault injection. Suesskraut et al. [24] used VMs to speed FI campaigns by taking a snapshot of the full execution state before an experiment and then rolling back to the pre-injection state. This also allows for all software dependencies to be fully contained within the guest VM to allow tests to be spread across multiple physical machines. This encapsulation of the experimental environment was noted by Clark et al. [6] as a benefit for reproducing results and performing repeatable research.

DeBardeleben et. al [7, 9] have used virtualization to develop a platform for vulnerability assessments. Their approach is based on the widely used QEMU emulator, which supports a dynamic translation layer for evaluating the instructions executed by the guest VM. Their tool, *F-SEFI*, can be used to study the effect of soft errors on applications. They have used the tools to simulate soft errors to affect instruction operands (e.g., corruption of operands to `FMUL` instruction), which can be done randomly or on a per-function basis for an application. They model soft errors as single or multi-bit corruptions and can inject the errors on a deterministic and probabilistic basis. This work uses a different virtualization environment (QEMU) from our type-II virtualization software (Palacios). Also, they introduce errors at the instruction level via the dynamic translation layer of QEMU, whereas our approach introduces errors via a character device that exposes the guest's memory with VM introspection techniques to identify the full process and memory layout for the target environment.

Le and Tamir [14] highlight advantages and challenges associated with using virtualization for FI based on their experiences developing and using the *Gigan* tool. They studied the fidelity of software implemented fault injection (SWIFI) running injection campaigns in a virtualized context versus running without virtualization (i.e., on bare hardware) and found the environments are comparable with some clear benefits for SWIFI based studies, i.e., isolation, logging, fast boot and crash detection. Their Xen based tool, *Gigan*, employed fault injectors at the (a) VMM level for injecting from outside the guest VM, and (b) kernel level for targeting kernel-space data structures and user-space processes within the VM. Lastly, they used the *Gigan* FI tool to develop a more robust hypervisor (ReHype) [14].

Note, others [5, 22] have investigated the fidelity of SWIFI in comparison to other FI approaches, showing that in some instances the software-based approach may be susceptible to an overestimation of errors in contrast to non-SWIFI approaches. The lessons being that single-bit failures introduced via SWIFI at the program level (in contrast to RTL or environment/hardware) may overestimate the effects of bit-flips. This has a bearing on vulnerability analysis that is derived from synthetic injection campaigns. Koopman [12] cited similar concerns for avoiding pitfalls when using fault-injection as a basis for dependability benchmarking. Therefore, the mechanisms employed in our work may not accurately mirror true hardware vulnerabilities, but have use for application testing and controlled experimentation where the user is mindful of the potential overestimations associated with SWIFI.

Li et al. [15] developed a binary instrumentation fault injection tool for studying soft errors in HPC applications. Their tool, *BIFIT*, is based on the PIN instrumentation tool and includes failure characterization based on injections into specific symbols/data-structures in the target HPC application based on profiling information for the applications. This work did not employ virtualization, but did study the effects of simulated "soft errors" on three HPC applications (Nek50000, S3D & GTC) by injecting bit-flips into global, heap and stack data objects. They limited the injections to application specific data, i.e., exclude middle-ware libraries, and observed that global data was significant to the influence of all three application's output and execution state. They also observed time and location of the injection is significant for each application with injections at later stages of application execution seeming to have a greater influence on the application's output & execution state. These soft error injections also affected the execution duration (walltime) of these applications, often with a 2x or greater increase in execution time. In our experiments, we target a different HPC application but focused on application specific data that is algorithmically important.

3 Cooperative Approach to Fault Injection

The placement of the SUT in a virtual machine enables FI campaigns to maintain a separation between the target and controlling system, regardless of whether the victim resides in user or kernel space. The separation of virtual/physical resources allows the resilience tests to operate within a *guest* virtual machine. The FI tests can be run from within the VM or from the *host* level entirely outside of the guest context. This division permits the host to control the guest, and can be an opportunity to modify the

state within the guest (e.g., inject virtual device errors, inject data corruption into guest memory). This separation does increase the complexity involved in the experimental environment and requires additional steps to overcome the semantic gap between the host and guest contexts.

3.1 Fault Injection Mechanism

The FI mechanism is implemented using a modification to the Palacios VMM that exports the guest VM's memory as a character device in the host OS. This device file enables host-level access to the memory of the guest OS and user-space tasks. A VM FI utility (VMFI) that runs on the host is configured with details about primary data-structures of the guest OS, e.g., address of the task structure symbol `init_task`. This provides details about the context of the kernel running in the guest VM and is similar to techniques used for VMI [17].

In the guest OS, another utility is used to provide a well-known marker to search for within the list of tasks. This is a small launch utility called *wrapper* that simply starts a command, i.e, `fork()/exec()`. This `wrapper` command is used to identify the process to target within the guest context.

On startup the `wrapper` utility prints its process identifier (PID). This PID can be passed as input to the VMFI utility, running outside of the VM, or the VMFI utility can be used to scan for the `wrapper` process in the VM. In the case of scanning for the `wrapper` process, the list of tasks within the guest OS is traversed (from outside the guest OS) to find all instances where the process name matches and the associated PID is displayed. This information (PID) then provides the necessary pointers to obtain the children tasks started by the `wrapper` and details about memory associated with those children. This lookup procedure results in the VMFI utility knowing the location of the memory associated with the `wrapper`'s child process, which is the target (victim) application running in the guest OS.

The startup of the `wrapper` and `vmfi` are currently manual steps. The other critical data that is necessary for the VMFI utility to function correctly is: the symbol names and addresses for the target application (that will run within the guest OS), and the value to write to the victim's target address. These target addresses are limited to symbol names in order to simplify the lookup process. The value to inject is provided as input to the VMFI utility. A brief description with example usage information for the VMFI and `wrapper` utilities is given in Figures 1 & 2.

4 Evaluation

When performing fault injection experiments the integrity of the target environment can be corrupted and lead to unexpected behavior. The use of virtualization provides a software layer that strengthens the isolation between the guest (target) and host (control). The following tests were performed to demonstrate the cooperative approach to VM-based experiments that use guest system and application context running in the VM to perform fault injection from the host environment (outside the VM). While not tested here, the VM-based FI approach can be used for tests targetting system software in the

```
1   Usage:
2    ./wrapper <executable> [args]
3
4   Description:
5    Wrapper utility to launch application and display useful information.
6    Also, used a sentinel for locating the target process in the guest
7    context, which is the child process of the wrapper utility.
8
9   Example:
10    ./wrapper ./HPCCG 100 200 100
```

Fig. 1. Usage information for wrapper utility that runs within the guest VM context.

```
1   Usage:
2   ./kmem list
3     --or--
4   ./kmem <wrapper_pid> <wrapper_map_file> <victim_map_file>     \
5          <target_symbol> <data_to_inject> <data_num_bytes>      \
6          <offset_from_symbol>
7
8    wrapper_pid             The pid of wrapper process residing in the guest
9    wrapper_map_file        Mapping file for the wrapper process
10   victim_map_file         Mapping file for the victim process
11   target_symbol           The name of the symbol in the victim
12                             process to inject a fault
13   data_to_inject          What to inject into the victim process
14   data_num_bytes          How many bytes to write
15   offset_from_symbol      Any additional bytes (offset) from target symbol
16
17  Description:
18    VMFI utility that can be used to LIST information about the guest context,
19    or used to inject errors into a victim application running in the guest
20    context.
21
22  Example:
23    ./kmem 198 wrapper.map HPCCG.symmap rtrans 6 4 0
```

Fig. 2. Usage information for VMFI utility that runs on the host (outside VM).

VM that operates in a privileged mode and could crash or misbehave, without affecting the controller on the host.

4.1 Setup

The experiment used the Palacios VMM running within a Linux v3.5.0 host OS. The guest OS is a Linux v2.6.33.7 kernel using Busybox v1.20 to create a very small system installation. The guest VM configuration included shadow memory paging. The guest used bridged networking, whereby a Linux virtio network interface in the guest was connected to the host's network interface. The HPCCG: Simple Conjugate Gradient Benchmark [16] was used as the target application. All tests were performed on a Linux cluster testbed (*SAL9000*) at ORNL. The machines in the cluster have 1 AMD64 CPU with 24 cores, 64 GB of memory, and dual-bonded 1 Gbps Ethernet. The host operating system was Ubuntu Linux 12.04 LTS.

4.2 Guest Application Errors

To investigate the feasibility of doing host-level injections into a guest-level context, the FI mechanism for Palacios described in Section 3 was leveraged. The HPCCG benchmark was used to test this FI functionality. The benchmark performs an iterative refinement until reaching a solution within a given threshold, or until a maximum number of iterations are performed. Previous studies have found iterative algorithms to be resilient to some errors [3], possibly at the cost of taking longer to converge on an appropriate value. The HPCCG benchmark has also been identified as a more representative metric for current scientific applications and was identified by Heroux and Dongarra as an alternate metric for future Top 500 indexes [10]. The HPCCG benchmark was slightly modified to expose the `rtrans` variable in the `HPCCG` function to be a global symbol. This was necessary in order for the `vmfi` utility to locate a target address within the guest OS. The `rtrans` variable was selected through manual code inspection; the variable is used throughout the life of the iterative application. The only other change to HPCCG was to vary the value of `tolerance` to allow the algorithm to adjust the solution threshold. For example, `tolerance=0.0` results in the algorithm always running to the maximum number of iterations [16], in contrast to setting `tolerance=0.0000001` that allows for a slight margin that can satisfy the threshold and (possibly) terminate before the maximum number of iterations. The binary was statically linked and run in serial mode (i.e., no use of MPI or OpenMP).

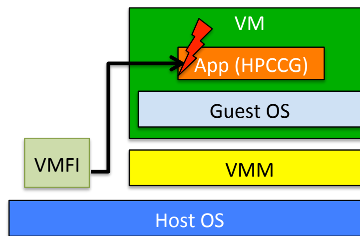

Fig. 3. Diagram showing the VM+FI setup with an application (e.g, HPCCG) target running in the Guest VM.

The overall layout is shown in Figure 3. The host level `vmfi` injects a value into a specified memory address within the context of an application running within the VM. The application used in our tests, HPCCG, is reflected by the orange *App (HPCCG)* box that resides in the space of the VM (green box). Figure 3 also illustrates the `vmfi` utility running outside the VM context and injecting an error into the target running within the VM.

4.3 Discussion & Observations

The guest application error testing confirmed that the host-level injector functioned correctly and caused non-fatal errors in the target application, HPCCG. The intent was

to simulate, at a very course-grain, data corruption of a key variable in the HPCCG program. The application was run 30 times both with and without injected errors. The same input parameters were used for all runs, $nx = 100$, $ny = 200$, $nz = 100$, which are the blocks of the matrix in the x/y/z dimensions [16], e.g., `wrapper ./test_HPCCG_tol0.0 100 200 100`. These values were selected to fit the available memory size and keep the execution time for the benchmark within the VM to a small amount of time to speed testing. The default maximum iterations `max_iter=150` was used, and the tolerance was set to `tolerance=0.0000001`. All non-error cases resulted in identical output for the value of the *Residual* (`rtrans`) on each iteration, and the *Final residual* printed at the end (`normr`) as shown in Table 1(a). The same tests were re-run with errors injected into the `rtrans` variable during the execution. The fault injections took place at 1 second intervals and injected a random value between 1..100. This value was written as 4-bytes into the target variable (`rtrans`) to emulate multiple bit-flips in a single data value. As expected, there were no fatal errors as the changes were controlled to be only in the specific data value of `rtrans`, but there were slight perturbations due to the data errors as shown by Table 1(b) which did not occur in the non-error case of the benchmark. This experiment verified the ability to perform silent data corruption into an application running in a guest OS context from the host OS. All tests (with and without errors) completed in 74 iterations.

Table 1. The effects to the *Final residual* (`normr`). These statistics show the results for the serial HPCCG test without (a) and with (b) random data errors. In the error case, values between [1..100] at 1 second intervals were injected into the `rtrans` variable. The statistics are based on the *Final residual* at the end of the benchmark. The parameters for the benchmark were $nx = 100$, $ny = 200$, $nz = 100$, and $tolerance = 0.0000001$.

(a) HPCCG No Errors

Field	Value
Num. runs	30
Minimum	8.97885e-08
Maximum	8.97885e-08
Mean	8.97885e-08
Median	8.97885e-08
Mode	8.97885e-08
Variance	0
Std.Dev.	0

(b) HPCCG With Errors

Field	Value
Num. runs	30
Minimum	8.97878e-08
Maximum	8.97881e-08
Mean	8.97879766666667e-08
Median	8.9788e-08
Mode	8.9788e-08
Variance	8.0500807188788e-27
Std.Dev.	8.97222420522292e-14

5 Conclusion

The use of VMs offers the ability to strongly separate the target from the hosting environment, which is useful when conducting fault injection experiments. The hosting platform has full access to the virtual guest context, but the details within the guest VM are not transparent from outside the guest's context. To overcome this issue a cooperative approach was explored where details about the guest OS were made available

to tools in the host context. In the guest context, additional `wrapper` command was added that provides information that host level tools can be leveraged to lookup details within the guest context. Additionally, the symbol maps for the guest kernel and application were made available to the host-level VMFI tools. This cooperative approach helps to reduce the semantic gap between the VM/host contexts.

The VM also provides a reusable execution context to support repeatable test configurations. This is very useful when creating a cooperative testing environment because the guest configuration is well known and customized as appropriate. Therefore assumptions can be made for the purposes of the FI experiments. For example, precompiled binaries can be placed in the VM that are also available on the host so symbol information (name/address) can be used for the FI experiments. This holds for the guest OS too, which can be made available at the host level for performing experiments on guest kernel data structures (e.g., via embedded VMM debuggers) or for accessing information about processes within the guest OS. The key insight being that the VM offers a customizable container that can be adapted as needed to simplify and aid FI experiments. The VM also offers full access to the guest context that would otherwise be difficult to achieve from a purely software approach.

A disadvantage of this low-level VMFI approach is an increased level of complexity and an increased semantic gap. This gap emerges because the higher level contextual information about the application (target) is divorced from the lower level VM vantage point. To overcome this challenge additional capabilities may need to be put in place, i.e., cooperative services, that provide additional information about the application context. For example, while the memory region for a guest OS is known by the VMM, the guest OS specific data structures within the VM are opaque. Therefore, a cooperative exchange of data is necessary to inform the host about details associated with the guest OS. For example, providing the VMM with a system map with the symbol names and address of functions and data structures of the guest OS running within the VM.

The prototype VMFI approach that we discussed in this paper was greatly influenced by VMI techniques. As demonstrated in the experiment, we were able to use these techniques to inject errors from outside the VM into specific data structures of a real benchmark (HPCCG) running within the guest VM. The iterative solver (HPCCG) reached the correct result, as expected, but the effects of our silent data corruption were detectable in an increased variance in the final residual (`normr`). While this experiment is very simplistic, it does show that the VMFI tool is working correctly and is usable for studies on applications running within a VM.

This work used the strong isolation of VMs to separate the FI controller from the FI target. Another approach that would be interesting to explore is the use of container-based virtualization to provide the isolation between the FI controller and target. The failure isolation properties of VMs and containers are not identical, and the container-based environments are restricted to a single OS kernel. Therefore, if the intent was to pursue FI campaigns against low-level system software (e.g., guest OS targets), the VMFI approach would be a better option than a container-based approach. However, if the target is an entirely user-space application, the isolation between containers may be sufficient for the FI experiments. A container-based approach would not suffer the semantic gap problem associated with VMs because there is a single OS kernel and

the FI controller (outside container) could have full visibility of all running processes. In general, further investigation is required to better understand the failure isolation properties of these single and multiple kernel approaches to virtualization.

6 Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research program.

References

1. Aderholdt, F., Ghafoor, S., Siraj, A., Scott, S.L.: Integrity based intrusion detection system for enterprise and cloud environments. In: Proceedings of the 4th IEEE/ACM International Conference on Utility and Cloud Computing (2011)
2. Brightwell, R., Oldfield, R., Maccabe, A.B., Bernholdt, D.E.: Hobbes: Composition and virtualization as the foundations of an extreme-scale OS/R. In: Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers (ROSS). pp. 2:1–2:8. ROSS'13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2491661.2481427
3. Bronevetsky, G., de Supinski, B.: Soft error vulnerability of iterative linear algebra methods. In: Proceedings of the 22nd Annual International Conference on Supercomputing. pp. 155–164. ICS '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1375527.1375552
4. Chen, P.M., Noble, B.D.: When virtual is better than real. In: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems. pp. 133–138. HOTOS '01, IEEE Computer Society, Washington, DC, USA (May 2001), http://dl.acm.org/citation.cfm?id=874075.876409
5. Cho, H., Mirkhani, S., Cher, C.Y., Abraham, J.A., Mitra, S.: Quantitative evaluation of soft error injection techniques for robust system design. In: Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE. pp. 1–10 (May 2013)
6. Clark, B., Deshane, T., Dow, E., Evanchik, S., Finlayson, M., Herne, J., Matthews, J.N.: Xen and the art of repeated research. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference. pp. 47–47. ATEC '04, USENIX Association, Berkeley, CA, USA (2004), http://dl.acm.org/citation.cfm?id=1247415.1247462
7. DeBardeleben, N., Blanchard, S., Guan, Q., Zhang, Z., Fu, S.: Experimental framework for injecting logic errors in a virtual machine to profile applications for soft error resilience. In: Proceedings of the 2011 International Conference on Parallel Processing - Volume 2. pp. 282–291. Euro-Par'11, Springer-Verlag, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-29740-3_32
8. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proc. Network and Distributed Systems Security Symposium (February 2003)
9. Guan, Q., Debardeleben, N., Blanchard, S., Fu, S.: F-sefi: A fine-grained soft error fault injection tool for profiling application vulnerability. In: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. pp. 1245–1254 (May 2014)
10. Heroux, M.A., Dongarra, J.: Toward a New Metric for Ranking High Performance Computing Systems. Tech. Rep. SAND2013-4744, Sandia National Laboratories (Jun 2013), http://www.sandia.gov/~maherou/docs/HPCG-Benchmark.

pdf, uRL: http://www.sandia.gov/˜maherou/docs/HPCG-Benchmark.pdf (Last visited: April 26, 2014).
11. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault injection techniques and tools. Computer 30(4), 75–82 (Apr 1997)
12. Koopman, P.: What's wrong with fault injection as a benchmarking tool? In: Proceedings of the Workshop on Dependability Benchmarking (WDB'02) (Jun 25, 2002), http://homepages.laas.fr/kanoun/ifip_wg_10_4_sigdeb/external/02-06-25/index.html, in conjunction with IEEE Conference on Dependable Systems and Networks (DSN-2002)
13. Lange, J., Pedretti, K., Hudson, T., Dinda, P., Cui, Z., Xia, L., Bridges, P., Gocke, A., Jaconette, S., Levenhagen, M., Brightwell, R.: Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing. In: IEEE International Symposium on Parallel Distributed Processing (IPDPS). pp. 1–12 (Apr 2010)
14. Le, M., Tamir, Y.: Fault Injection in Virtualized Systems – Challenges and Applications. Transactions on Dependable and Secure Computing 12(3), 284–297 (May/June 2015), http://www.cs.ucla.edu/~tamir/papers/tdsc15.pdf
15. Li, D., Vetter, J.S., Yu, W.: Classifying soft error vulnerabilities in extreme-scale scientific applications using a binary instrumentation tool. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC). ACM (Nov 2012)
16. Mantevo mini-application downloads, http://www.mantevo.org/packages.php, project URL: http://www.mantevo.org/packages.php (Last visited: April 6, 2014).
17. Nance, K., Bishop, M., Hay, B.: Virtual machine introspection: Observation or interference? IEEE Security and Privacy 6(5), 32–37 (2008)
18. Naughton, T., Vallée, G., Engelmann, C., Scott, S.L.: A case for virtual machine based fault injection in a high-performance computing environment. In: Proc. of the 5th Workshop on System-level Virtualization for High Performance Computing (HPCVirt'11). pp. 234–243. Euro-Par'11, Springer-Verlag (2012)
19. Payne, B.D., Carbone, M., Sharif, M., Lee, W.: Lares: An architecture for secure active monitoring using virtualization. In: Proceedings of the IEEE Symposium on Security and Privacy (May 2008)
20. Popek, G.J., Goldberg, R.P.: Formal requirements for virtualizable third generation architectures. Commun. ACM 17(7), 412–421 (1974)
21. Potyra, S., Sieh, V., Cin, M.D.: Evaluating fault-tolerant system designs using FAUmachine. In: Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems (EFTS'07). p. 9. ACM, New York, NY, USA (2007)
22. Schirmeier, H., Borchert, C., Spinczyk, O.: Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 319–330 (June 2015)
23. Smith, J.E., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann (2005)
24. Süßkraut, M., Creutz, S., Fetzer, C.: Fast fault injection with virtual machines (Fast Abstract). In: Supplement of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN2007) (Jun 2007), http://wwwse.inf.tu-dresden.de/papers/preprint-suesskraut2007DSNb.pdf