A Case for Virtual Machine Based Fault Injection in a High-Performance Computing Environment*

Thomas Naughton, Geoffroy Vallée, Christian Engelmann, and Stephen L. Scott

Oak Ridge National Laboratory,
Computer Science and Mathematics Division,
Oak Ridge, TN 37831, USA

Abstract. Large-scale computing platforms provide tremendous capabilities for scientific discovery. As applications and system software scale up to multi-petaflops and beyond to exascale platforms, the occurrence of failure will be much more common. This has given rise to a push in fault-tolerance and resilience research for high-performance computing (HPC) systems. This includes work on log analysis to identify types of failures, enhancements to the Message Passing Interface (MPI) to incorporate fault awareness, and a variety of fault tolerance mechanisms that span redundant computation, algorithm based fault tolerance, and advanced checkpoint/restart techniques.

While there is much work to be done on the FT/Resilience mechanisms for such large-scale systems, there is also a profound gap in the tools for experimentation. This gap is compounded by the fact that HPC environments have stringent performance requirements and are often highly customized. The tool chain for these systems are often tailored for the platform and the operating environments typically contain many site/machine specific enhancements. Therefore, it is desirable to maintain a consistent execution environment to minimize end-user (scientist) interruption.

The work on system-level virtualization for HPC system offers a unique opportunity to maintain a consistent execution environment via a virtual machine (VM). Recent work on virtualization for HPC has shown that low-overhead, high performance systems can be realized [7,15]. Virtualization also provides a clean abstraction for building experimental tools for investigation into the effects of failures in HPC and the related research on FT/Resilience mechanisms and policies. In this paper we discuss the motivation for tools to perform fault injection in an HPC context. We also present the design of a new fault injection framework that can leverage virtualization.

1 Introduction

Large-scale computing platforms provide tremendous capabilities for scientific discovery. These systems have hundreds of thousands of compute cores, hundreds of terabytes of memory, and enormous high-performance interconnection networks. As these platforms increase in size to accommodate the performance demands of the computational

* ORNL's work was supported by the U.S. Department of Energy, under Contract DE-AC05-00OR22725.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 234–243, 2012.

science community, their component counts and overall system complexity increase. These massive node/component counts also yield increased occurrence of failure. As such, fault tolerance/resilience (FT/R) are key factors for effective utilization of high-performance computing (HPC) systems.

Going forward, failures will become something that developers and scientific users of these HPC platforms will be forced to manage in their applications as they move toward multi-petaflop and exascale systems. There is already active work on this topic, to include log analysis for identification of failure types/modes, enhancements to the Message Passing Interface (MPI) to incorporate fault awareness, and a variety of fault tolerance mechanisms that span redundant computation, algorithm based fault tolerance, and advanced checkpoint/restart techniques.

However, there are few tools to aid in the evaluation and experimentation of HPC failures. The methodical investigation of failure in these systems is hampered by their scale, and a lack of tools for controlled experiments. The area of system-level virtualization offers a promising basis to support such experimentation. Virtualization has received much attention in recent years, which was primarily driven by commercial efforts (e.g., server consolidation), such that vendors are including hardware support for virtual machines. This trend and wide-scale industry adoption is likely to lead to virtualization as a standard capability of modern hardware/operating systems.

As the importance of fault tolerance increases, methods for experimentation into new mechanisms and policies is critical. The widespread availability of virtualization, combined with its strong isolation and user-customizable/adaptable execution environments, makes it an interesting platform for fault-tolerance testbeds. The virtual machine allows various operating systems, middleware, and applications to run unmodified in a controlled environment, where dependability can be evaluated at a very generic level, e.g., via virtual machine based fault injection.

Motivated by work in FT/R for HPC and current efforts in virtualization, we argue that tools for FT/R experimentation are needed and that system-level virtualization provides an interesting basis to develop such tools. We highlight prior work into the use of fault-injection for FT/R evaluation. We also discuss some of the unique requirements that emerge when working in an HPC environment.

The remainder of the document is organized as follows: Section 2 presents existing solutions for fault injection that are representative of the current state of the art, as well as a classification of these different solutions; Section 3 motivates the work presented in this document and Section 4 presents the specifics of the HPC context. Section 5 proposes a new framework architecture for fault injection in the context of HPC, and Section 6 concludes.

2 Background

Fault injection is the purposeful introduction of faults (or errors) into a target [5]. It has been used extensively for testing and experimentation of fault-tolerance mechanisms. These injections may be via specialized hardware or through software implemented fault injection (SWIFI). The software based approach offers more flexibility in terms of how to implement and detect the faults, but are limited in scope to areas accessible

via software [5]. For example, radiation induced memory "soft errors" can be injected via hardware/environmental approaches but can only be emulated through software by techniques like bit-wise operations to force memory bit-flip(s).

The past work has included testing for distributed environments (NFTAPE) [12], which involved some experiments with MPI applications. In their MPI tests, they made minor modifications to the target application to accommodate the cluster based launch mechanism. In more recent work, FAult Injection Language (FAIL), provides a user the ability to express fault injection scenarios in a high-level language for distributed systems [4]. The language includes a compiler and execution context for performing the distributed fault-injection campaigns. The initial FAIL framework used the FCI (FAIL Cluster Implementation) to perform remote startup. In later work [3], they extended the tool to support dependability benchmarking of a fault-tolerant implementation of MPI.

Earlier studies have looked at the effects of faults on parallel applications running on a 4 node Parsytec PowerXplorer Transputer based machine [11]. This work leveraged hardware capabilities, e.g., performance counters and debug registers, to create low-overhead fault injection mechanisms (Xception) [6]. This provided a method to inject faults into any process, to include system software.

In systems that support dynamic loading of shared libraries, a common approach is to use the `LD_PRELOAD` mechanism to interpose on library calls and introduce faults, e.g., library level fault injection (LFI) [9]. This is a generic approach that does not require source code modifications of the target, and allows for runtime additions via the shared library wrapper routines.

Virtual machines have been used to aid in application robustness testing. In "FastFI with VMs" [13], they used virtual machines to aid in fault injection, specifically to reduce the complexity in capturing state associated with an application under test. They inject faults within the VM (e.g., API robustness for system calls from application), and export the post-fault analysis results and experimental state to the host before a VM rollback. Their focus is on using this efficient shapshot/rollback to reduce the overall runtime for extensive fault-injection campaigns. They mention the challenges with capturing state for fast rollback when working without VMs (their prior work employed `fork()`) and how some experiments would result in residual effects from the FI post-rollback, which was avoided with the isolation/encapsulation of the VM. They mention that distributed applications can be tested by using VM's, and virtualizing the network by running all on a single host. However, it appears they restrict their use of VM's and FI to a single physical machine, even in the case of testing distributed applications. Additionally, their work was focused on software error handling using fault injection at the API level.

The FAUmachine [10] project has produced a hardware simulation platform that provides fine grained fault injection capabilities. They use VHDL to model the hardware and leverage this detail to create more realistic SWIFI scenarios. They report that running unmodified operating systems and applications on the virtual CPU of FAUmachine, with its just-in-time compiler, results in a 5-20% reduction in performance in comparison to native execution. Their system supports a variety of faults, to include [10]: CPU, IDE controller, network adapter, serial terminal, monitor, power, and keyboard/mouse (input) errors.

A stated goal of the FAUmachine project is to maintain the fidelity of the hardware, as opposed to other virtualization efforts which seek increased performance in spite of true execution behavior. Restated, FAUmachine attempts to keep the hardware details consistent with real actions, whereas other virtual machine based system seek to improve performance and may adjust execution to achieve this goal.

In [8] the advantages and challenges associated with using virtualization for SWIFI are discussed. They cite the ability to isolate the system under test and avoid cumbersome situations when injecting into system running directly on the physical machine (e.g., self-crashes, log corruption). The challenge when working with VMs are to maintain sufficient fidelity with standard (bare machine) execution when performing injections, i.e., inject into proper context and avoid indirect virtualization related side effects. They use their tool, *Gigan*, to perform fault injection from within the VM (guest kernel) and from the VMM, for both para-virtualized and fully virtualized VMs. They tested a non-virtualized instance (where guest would normally run on physical machine) and virtualized case (to test resilience of VM and VMM platform itself).

Table 1. The projects discussed in this background section fall into three groups: (i) tools for distributed systems, (ii) mechanisms that leverage hardware/software features, and (iii) FI systems that employ virtualization

Distributed Systems	Leverage HW/SW Features	Virtualization
FAIL-FCI/-MPI	LFI	FAUmachine
NFTAPE	Xception	FastFI-VM
		Gigan

3 Motivation

Testing and experimentation are fundamental components of computer research and development. As systems increase in size this process becomes increasing difficult due to factors like distributed resources and concurrent execution. Additionally, it is becoming increasingly apparent that large-scale computing platforms must cope with an increased occurrence of failures. While this aspect has been embraced in purely distributed environments, it has been less common in high-performance computing (HPC) due to a more optimistic view on failures for these tightly coupled platforms. This leads to a growing recognition that failures are an unavoidable reality in large-scale high-performance computing (HPC) systems. Therefore, the system software that underpins these HPC systems must evolve to provide better fault tolerance (FT). Additionally, the applications themselves must also cope with failures and make effective use of any/all available FT mechanisms. An apparent indication of this merger of FT and HPC is the recent message passing interface (MPI) v3 working group on fault tolerance, MPI3-FT. Another indication that current and future large-scale systems must find new methods to cope with these interruptions is high-lighted in the recent DoE Exascale workshops and associated proposal calls. The objective being to design and implement the next generation software stack for future exascale computing platforms.

As fault-tolerance continues to push the research and development in HPC system software, the tools and techniques to properly test proposed solutions becomes more critical. Therefore, a systematic method for experimentation into fault-tolerance is necessary to help in the development of future HPC systems. The focus of our work is to provide such tools and create experimental environments for continued research and development. To that end, we discuss requirements and challenges associated with buildings failure testbeds supported by system-level virtualization. The use of virtualization provides two key benefits: (i) leverages modern hypervisor technology to reduce the overhead of running a guest testing domain, and (ii) provides a consistent execution environment for the target software stack, which is isolated from the native/physical platform. This enables the fault-tolerance efforts to develop as needed, while maintaining a consistent base testing/experimentation platform.

4 HPC Environments

There are several constraints associated with tools for HPC systems. In this section we highlight some significant aspects involved in building tools for those environments, with the purpose being to indicate properties that may affect fault injection tools for HPC environments. Also, in Table 2 we provide a brief contrasting of the prior work mentioned in Section 2 with the HPC oriented properties discussed in this section.

4.1 Batch Allocation Systems

These large systems are generally run using some form of job management system where applications (jobs) are submitted to a batch scheduling system, e.g., PBS/Torque, SLURM. In batch scheduled environments the job is allocated a dedicated set of resources for a given period of time. However, access to those resources may be highly restricted. For example, process startup may be limited to a remote invocation interface with no direct (interactive) access granted to the compute resources. That is to say the user may only be able to startup a task, but may not actually have a local shell to each processing element (node). This requires the tool chain to support these remote execution interfaces. This may also have ramifications on how the actual fault-injection experiments are carried out or detection is performed on the remote processes.

4.2 Executable Linkage

The compute node execution environments on HPC systems may be highly specialized. There may not be a local hard drive on the node, and there may be a minimalistic runtime environment. This often constraints how the executable binary is compiled. In many instances, there are no shared libraries on the compute nodes or dynamic shared library loading support is disabled entirely. This may limit the applicability of existing fault injection frameworks for practical use in an HPC environment. For example, the compiled binaries may require static linkage for execution on the compute nodes. This precludes the direct use of many fault-injection techniques that leverage shared libraries to interpose on application execution, e.g., `LD_PRELOAD`.

4.3 Performance Sensitive

The tool chain must be as low-overhead as possible. This has noticeable effects on infrastructure that must scale to support large numbers of processes. The tools may be forces to avoid centralized approaches due to added load on the network communication links, which could affect the performance of the application. In the context of fault-injection, this may manifest as constraints on monitoring frequency and volume of reporting information. Additionally, the concept of "performability" may be relevant, where performance and availability are criteria in the evaluation of the given application. Since fault-tolerance and performance often share many parallels, these aspects are likely to be relevant in the fault-injection infrastructure driving FT/R experiments.

4.4 Exotic Hardware

The hardware for these systems may be rather "exotic", and therefore require additional insights for its use. This may be in the form of additional software interfaces or device drivers. This is often the case in the interconnection network of HPC systems, which may influence the communication API provided to the tools. The software infrastructure may also have to accommodate idiosyncrasies in the hardware.

Note, the difference in HPC hardware may also influence the ways faults occur, i.e., "fault types". This is very important when defining fault-injection experiments. The mapping of applications to the system resources will also influence how the system behaves in the presence of faults. For example, the HPC interconnection network has greater bandwidth along one axis. Then failures in this direction could generate congestion, which could result in clients failing in their writes to the parallel file system. The point being that these errors were influenced by characteristics of the hardware and expectations on system balance, which can lead to complex failure scenarios that are unique to these platforms.

Table 2. Summary of HPC oriented properties supported by FI projects from Section 2

a The distributed MPI test cases with NFTAPE required modification to the source to cope with cluster launch

HPC Oriented Properties	FAIL-FCI	NFTAPE	LFI	Xception	FAUmachine	FastFI-VM	Gigan
Target Source Modifications	N	Y_a	N	N	N	N	N
Distributed Environment	Y	Y	N	Y	N	N	N
Full Execution Environment	N	N	N	Y	Y	Y	Y
HPC System Interfaces	Y	N	N	N	N	N	N
Require Dynamic Linkage	N	N	Y	N	N	N	N

5 Design

Our approach is to leverage existing work in HPC system software to build a framework for testing the effects of failure. By leveraging existing HPC infrastructure we can better manage some of the issues outlined earlier in Section 4. The major components of the system can be broken down into the following areas: (i) front-end and distributed control, (ii) experiment setup/management, (iii) monitoring and event logging, and (iv) fault injection mechanisms. The user interacts with the framework via the front-end to drive an experiment to test the effects of failure on a target application. The basic structure of the system is shown in Figure 1. The remainder of this section describes the aspects of these areas, to include what parts of the existing system software will be leveraged and/or extended.

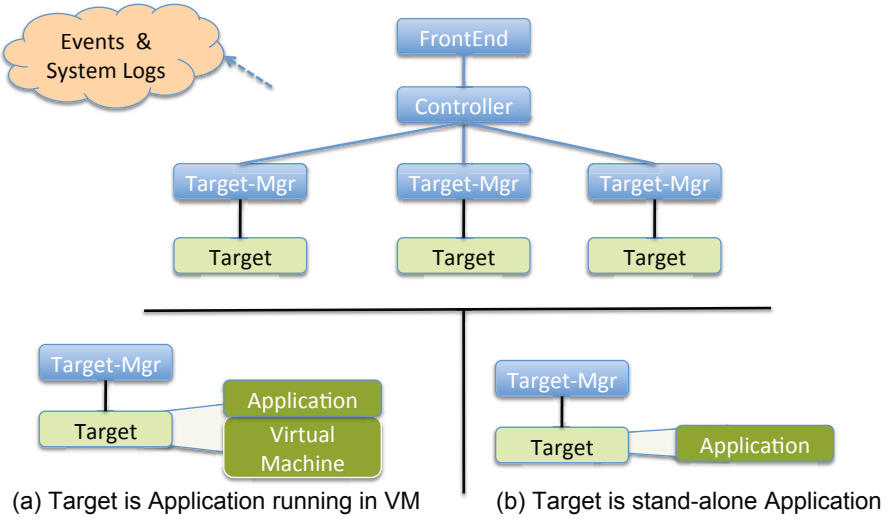

Fig. 1. Basic structure for the tool

5.1 Front-End and Distributed Control

The interface provided to users of HPC systems is generally in the form of a launcher that the user invokes, which in turn invokes platform interfaces for remote task startup and control. This runtime environment (RTE) provides a basic communication and distributed control layer that applications may use to interact. The message passing interface (MPI) programming model is a common example that will be familiar to most users of HPC systems. The RTE provides MPI applications a way to bootstrap tasks and their communication handles. This is the same kind of functionality that is needed to startup and manage our fault injection experiments. Therefore, we leverage the RTE code base of the Scalable Tools and Communication Infrastructure (STCI) project [1]. This is a component-based RTE platform being developed at Oak Ridge National Laboratory.

In Figure 1 the STCI related parts are delimited by the blue boxes. This includes a Frontend and Controller that provide the interface between the user and system respectively. A customized agent is created called a "Target Manager". This provides the interface between the control runtime and the actual experiment "Target". The Frontend, Controller and Target Manager share a common communication space provided by STCI. The Target Manager is responsible for the bootstrap and control of the "Target", which will be discussed in more detail below.

5.2 Experiment Setup/Management

A fault injection experiment includes a configuration that defines what events (faults) should be introduced and how the experiment should be carried out. The steps to carry out this experiment are managed by the testing framework. The system under test (SUT), or "Target", is the entity upon which the experiment is focused. The *Target* is paired with a *Target Manager*. This allows for a clean separation of the SUT from the infrastructure used to manage the experiment. In the context of our framework, the Target could be an application binary or a virtual machine with the application running inside the VM. This has an effect on what injectors and faults may be employed for the experiment, but these details should be hidden from the end-user as much as possible.

In the case where the Target involves virtualization, the Target Manager is responsible for VM bootstrapping and management. This VM interface leverages prior work for Virtual System Environments [14], specifically the *LibV3M* abstraction layer. This library provides a VM management interface that the Target Manager uses to start, stop and monitor the VM based experiment.

5.3 Monitoring and Event Logging

While experiments are taking place, the system should be monitored using existing tools like Ganglia, syslog analyzers, etc. Additionally, a specific event notification channel is established so faults may be published as they are injected and used to correlate with actual failures detected throughout the system. The STCI runtime provides basic monitoring and failure detectors that could be useful for experiment monitoring. The publish/subscribe services in STCI and CiFTS Fault Tolerant Backplane (FTB) [2] are also good candidates for implementing these event notification channels.

5.4 Fault Injection Mechanisms

The controlled injection of synthetic faults, i.e., fault-injection (FI), requires that the system under test be sufficiently isolated. The strong isolation capabilities of virtual machines provide a sound platform for this requirement. It is beneficial to keep as much, if not all, of the FI infrastructure outside of the VM in order to maintain these isolation properties. Therefore extending the VM interface to support FI capabilities allows for access from the host operating system, and avoids complications when performing the injections in the same execution environment as the FI infrastructure.

There are a number of approaches to implement fault injection mechanisms. Two key characteristics of the mechanisms are: (a) how they are invoked, and (b) where the fault should be introduced.

The invocation may be based on a timer or some other external trigger (i.e., user input). It could also be based upon access to some resource, i.e., read from a given memory address. This leads to the second facet, fault location. In the case of VM-based injectors, the VMM/VM have full access to the resources and have full control to interpose on their access. However, to make the experiments more meaningful, often details about the application running in the VM must be exported to improve context for fault placement. This is a significant part of the experiment procedure, and something that will require careful consideration during experiment design. The current work in the V3VEE project has added initial support to Palacios for generating exceptions (faults) for specific memory locations at the VM-level, which are then serviced by the guest operating system. The FI framework would provide a way to configure tests that could leverage this to perform injections over a set of machines.

6 Conclusion

As HPC systems increase in size, concerns about failures and application resilience also increase. There is a growing effort to explore techniques to provide improved fault-tolerance/resilience (FT/R) for HPC platforms. This includes research and development at both the system and applications levels of the software stack. However, there is a significant gap in the set of tools available to assist in experimentation for FT/R mechanisms and policies. In this paper we have provided motivation for tools to address this gap and comments as to why existing tools for fault-injection experiments may face challenges for reuse in an HPC context. We also described an initial architecture for a new fault injection framework, which is based on existing HPC system software and tools, to help overcome this gap. We also argue that current work in system-level virtualization provides a good basis for developing fault-injection tools for HPC due to its strong isolation capabilities and complete access to resources used by the application (via virtual machine abstraction).

References

1. Buntinas, D., Bosilica, G., Graham, R.L., Vallée, G., Watson, G.R.: A Scalable Tools Communication Infrastructure. In: Proceedings of the 22nd International High Performance Computing Symposium (HPCS 2008), June 9-11, session track: 6th Annual Symposium on OSCAR and HPC Cluster Systems (OSCAR 2008). IEEE Computer Society (2008), http://www.csm.ornl.gov/oscar08/
2. Gupta, R., Beckman, P., Park, B.H., Lusk, E., Hargrove, P., Geist, A., Lumsdaine, A., Dongarra, J.: Cifts: A coordinated infrastructure for fault-tolerant systems. In: International Conference on Parallel Processing, ICPP (2009)
3. Hoarau, W., Lemarinier, P., Herault, T., Rodriguez, E., Tixeuil, S., Cappello, F.: Fail-mpi: How fault-tolerant is fault-tolerant mpi? In: IEEE International Conference on Cluster Computing, pp. 1–10 (September 2006)

4. Hoarau, W., Tixeuil, S., Vauchelles, F.: Fail-fci: Versatile fault injection. Future Generation Computer Systems 23(7), 913–919 (2007), http://www.sciencedirect.com/science/article/pii/S0167739X07000209
5. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault injection techniques and tools. Computer 30(4), 75–82 (1997)
6. Carreira, J., Madeira, H., Silva, J.G.: Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers. IEEE Transactions on Software Engineering 24(2) (February 1998), http://www.xception.org/files/IEEETSE98.pdf
7. Lange, J., Pedretti, K., Hudson, T., Dinda, P., Cui, Z., Xia, L., Bridges, P., Jaconette, S., Levenhagen, M., Brightwell, R., Widener, P.: Palacios and Kitten: High Performance Operating Systems For Scalable Virtualized and Native Supercomputing. Tech. Rep. NWU-EECS-09-14, Northwestern University, July 20 (2009), http://v3vee.org/papers/NWU-EECS-09-14.pdf
8. Le, M., Gallagher, A., Tamir, Y.: Challenges and Opportunities with Fault Injection in Virtualized Systems. In: First International Workshop on Virtualization Performance: Analysis, Characterization, and Tools, Austin, Texas, USA (April 2008), http://www.cs.ucla.edu/~tamir/papers/vpact08.pdf
9. Marinescu, P.D., Candea, G.: LFI: A Practical and General Library-Level Fault Injector. In: Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009), June 29 - July 2. IEEE (2009), http://dslab.epfl.ch/pubs/lfi/index.html
10. 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 2007), p. 9. ACM, New York (2007)
11. Silva, J.G., Carreira, J., Madeira, H., Costa, D., Moreira, F.: Experimental assessment of parallel systems. In: Proceedings of the 26th Annual International Symposium on Fault-Tolerant Computing (FTCS 1996), June 25-27, pp. 415–424 (1996)
12. Stott, D.T., Floering, B., Burke, D., Kalbarczyk, Z., Iyer, R.K.: NFTAPE: A framework for assessing dependability in distributed systems with lightweight fault inectors. In: Proceedings of the 4th IEEE International Computer Performance and Dependability Symposium (IPDS), pp. 91–100. IEEE (March 2000)
13. 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) (June 2007), http://wwwse.inf.tu-dresden.de/papers/preprint-suesskraut2007DSNb.pdf
14. Vallée, G., Naughton, T., Scott, S.L.: System Management Software for Virtual Environments. In: Proceedings of the ACM International Conference on Computing Frontiers (CF 2007), Ischia, Italy, May 7-9 (2007)
15. Youseff, L., Seymour, K., You, H., Dongarra, J., Wolski, R.: The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC 2008), pp. 141–152. ACM, New York (2008)