HPC-SFI: System-Level Fault Injection for High Performance Computing Systems

Yanqi Wang, Qi Zhang, Yi Liu(✉), and Depei Qian

Sino-German Joint Software Institute,
Beihang University, Beijing 100191, China
wangyanqi0114@outlook.com, liuyi97@263.net

Abstract. Resilience/fault-tolerance has become a key challenge for large-scale parallel systems. To ensure reliability of high performance computing systems, various kinds of techniques have been proposed, such as hardware-level fault-tolerance, checkpointing, replication, algorithm-base fault-tolerance, etc. There are also many software systems to monitor and handle system-failures, e.g. management and job-scheduling system of HPC systems. To evaluate the effectiveness of these systems, it is necessary to provide some kind of tool to inject failures in a HPC system. This paper proposes HPC-SFI, a system-level fault injection tool for HPC systems. Basically, HPC-SFI can generate three kinds of system-failures in a HPC system including in-node faults, failure in the interconnection network and failure of storage/parallel-file system. In addition, HPC-SFI can inject system-faults in pseudo-random model according to pre-defined parameters and probabilities. Preliminary experimental results demonstrate effectiveness of the tool.

1 Introduction

With the scaling up of high performance computers in recent years, resilience, or fault-tolerance, has become a key challenge. Currently, top-ranking supercomputers generally have tens of thousands of processors, e.g. the Summit [1] has 8,712 processors and 26,136 GPUs, while the number of processors in the Sunway TaihuLight [1] is 40,960. Along with the increasing of system scale, hardware/software-failures occur more frequently. Statistics show that the MTBF (mean time between failure) of current most powerful supercomputers has reduced to several hours.

To ensure reliability of high performance computing systems, various kinds of techniques have been proposed, such as hardware-level fault-tolerance, checkpointing, replication, algorithm-base fault-tolerance, etc. There are also many software systems to monitor and handle system-failures, e.g. management and job-scheduling system of HPC systems. To evaluate the effectiveness of these systems, it's necessary to provide some kinds of tools to generate various kinds of failure in HPC systems. However, current fault injection tools either focus on injection of soft-errors and their influences over high-level applications, or inject system-level failure in emulated environments (e.g. virtual machines) to guarantee flexible control over the system.

This paper proposes HPC-SFI, a system-level fault injection tool for HPC systems. Unlike current fault injection tools, our HPC-SFI inject hardware/software-failures in real physical systems. Therefore is more suitable for machine vendors and developers of fault-tolerant-related system software to evaluate the effectiveness of their fault-tolerant mechanisms.

Main characteristics of HPC-SFI include:

(1) HPC-SFI can inject three kinds of failure to a HPC system: in-node faults, failure of the interconnection network and failure in storage/parallel-file-system. Typical in-node faults include processor halt, memory error, network interface/disk failure as well as the system halts.
(2) HPC-SFI cannot only inject deterministic failure in a HPC system, but also generate failure in pseudo-random model according to pre-set parameters and probabilities, which are more approximate to actual systems.
(3) The injected failure can be recovered after a predefined time period.
The rest of this paper is organized as follows. Section 2 discusses our methods of fault-injection; Sect. 3 introduces architecture of the system and implementation detail. Section 4 presents preliminary experimental results; Sect. 5 discusses related work and the paper is concluded in Sect. 6.

2 Approaches

2.1 Types of Failures

Possible hardware/software faults or failure in high performance computer systems are diverse. To make things simple, our HPC-SFI focuses on system-level failure, which means under this kind of failure, part or entirely of the system cannot work correctly. These failure either occur inside computing nodes, or outside the nodes, i.e. in interconnection network or storage system.

Based on the above discussion, our HPC-SFI considers three kinds of failures, described as follows:

(1) **In-node faults/failures**

In-node faults/failures can be further divided into faults/failures in a different component of the node, e.g. processors, memory, network interface card, etc. In addition, crash-down of an entire node should also be considered.

(2) **Failure of interconnection network**

This kind of failure either occurs in network cable or in switches. Obviously it will cause communication errors in multiple nodes.

(3) **Failure of storage or parallel file system**

Current high performance computers generally use dedicated storage systems together with parallel file systems to provide high-throughput I/O and shared storage to parallel applications running in different nodes, while the in-node hard-disk just used as system startup. Failure of storage or parallel file system may occur in various components of the storage system or dedicated I/O nodes, and will influence file-accesses of computing nodes.

2.2 Injection Methods

Different HPC systems have different hardware configurations, and generally come from different vendors. As a software tool, it is difficult for HPC-SFI to obtain controls over dedicated equipment such as interconnection switch or RAID array. In other words, some kinds of failures cannot be generated directly, as a substitution, we generate “effect” of the corresponding failure, e.g. failure of a switch will cause communication interruptions on all the nodes that connected to the switch, failure of storage or parallel file system will cause the corresponding file volume mounted to file systems of each node unable to access.

Table 1 shows phenomenon and injection methods of failure supported by HPC-SFI. As shown in the table, the in-node fault injection acts as the basis of the system, because the other two types of failures, the failure of interconnection network and storage system, are implemented upon the in-node fault injection, that is, inject failures in multiple specific nodes simultaneously.

Table 1. Failure phenomenon and injection methods

Kind of failures	Component	Phenomenon	Injection method
In-node fault/failure	Entire node	System halt	Halt the system by shell commands
	Processor	*1-n* processor (cores) stop working	Process forcibly consumes processor resources
	Memory	Contents of memory units error	Modify the specified memory content
	Network interface	Communication error	Disable HBA card by shell commands
	In-node fixed-disk	Disk error	Destroy the super block of the disk partition
	File volume access	Volume access error	Destroy the disk file resources
Failure of interconnection network		Communication interruptions in all of the related nodes	Disable HBA cards in all of the related nodes
Failure of storage or parallel file system		File volume access error in all of the related nodes	Destroy parallel file system

As for in-node fault injection, actually most of the in-node faults/failures can be generated using Linux shell commands except memory-fault injection, which is implemented in two forms: i. a kernel module which can access entire memory space; ii. a user-level interface which can be invoked by applications to modify its own data.

Another problem that needs to be solved is the recovery after the failure injection. Considering the system scale of current HPC systems, it is impractical to reboot each node after it is injected faults/failures, instead, the node must be recovered to its original state after a fault injection. Due to that most failures are generated using shell commands, it is easy to recover under the control of the system. The only one failure that need special treatment is the network-related failures, after the HBA card or NIC card is disabled, the node becomes isolated and cannot receive later recovering commands, in this situation, the node must be self-recovered, which is implemented using a shell script working in sequence of "*disable-delay-enable*".

2.3 Deterministic vs. Pseudo-random Fault Injection

To approximate actual failure in HPC systems, the HPC-SFI supports two fault-injection model: the deterministic fault-injection and pseudo-random fault-injection.

(1) Deterministic fault injection

Generate determined failures according to the specified parameters, such as a node, time, and the fault type.

(2) Pseudo-random fault injection

The fault probability is specified by setting the node range and the number of faulty nodes, the time range, and the fault type range. The fault can be generated according to the fault model, so as to simulate the actual running of HPC systems.

In actual HPC systems, the occurrence of faults/failures is non-deterministic and generally unpredictable. We define a four-tuple of fault-injection pseudo-random probability model for HPC systems to describe the probability of fault-injection execution:

$$P_{injec} = \langle T, R, NUM, F \rangle \tag{1}$$

Where *T* indicates that within a certain time range, *R* is the range of the nodes to be tested, *NUM* is the specified number of injection nodes, and *F* is the type range of the fault. The above parameters are defined, and a fault injection model is generated by a pseudo-random probability model. Such fault injection is more approximate to the occurrence of faults in real systems and can be used to evaluate the effectiveness of fault-tolerant diagnostics.

HPC-SFI tool failure is generated by the model, and the user can generate a fault parameter configuration by specifying a description file. In the configuration file provided by the tool, the user sets the relevant parameters according to requirements to define the four-tuple of fault injection probability model.
The process is as follows:

```
select NODE[R];
while NUM
  NUM --;
  rand(f);  rand(k);
  while Time
    inject FAULT[f] to NODE[k];
    Time --;
  end
end.
```

This program generates a corresponding fault injection model based on the fault description to implement fault injection control.

3 System Architecture and Implementation

Figure 1 shows the architecture of HPC-SFI. The HPC-SFI tool mainly consists of two parts: the first part is the *master* running in a control node; the second part is the *node-part* running on each node of the target HPC system.

In each node, an application process, named *HPC-SFI broker*, runs in the background waiting for commands from the *master*. On receiving a fault injection command, the *HPC-SFI broker* invokes the *In-node injection module* to generate corresponding faults/failures in the node. The *master* communicates with the *HPC-SFI brokers* via management network of the target HPC system. The *In-node injection module* can also be invoked by other application processes, at this time, the *master* is overridden and fault injection is controlled by the user-defined application.

In the *master*, the *model-generation module* parses the *failure description file* defined by the user, generates the fault parameter profile based on the description file, and uses these fault parameters to determine the type and time of a fault injection. In order to assess the effectiveness of fault injection, we measure reliability parameters such as test coverage and latency when performing the appropriate fault injection.

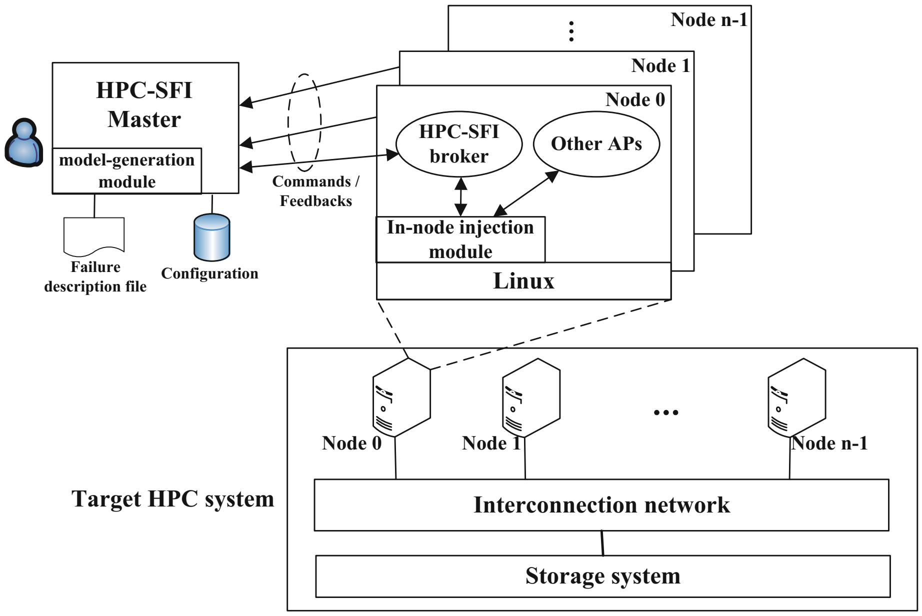

Fig. 1. Architecture of the system.

HPC-SFI realizes three types of failure of HPC system: in-node faults, failure of interconnection network, and failure of storage or parallel file system. Because the failure of the interconnection network and the failure of the storage parallel file system are generated based on the node fault, various faults of the traditional physical machine have been realized: memory faults, processor faults, network communication faults and disk faults.

Memory failure can be injected with a single bit or multiple byte error. The memory-fault injection program requires a scheme to modify the specified memory content, partially or completely setting. The virtual address of the process is converted into a physical address, and the process code segment data are directly modified in the memory according to the physical address, so as to achieve the purpose of fault injection. The Linux kernel module mechanism is introduced to obtain the privilege of modifying any specified memory location.

As mentioned in Sect. 2.2, failure of the interconnection network and storage system are implemented on the basis of in-node fault injection. For instance, when user specifies a switch-failure in the *fault description file*, the *model-generation module* parses the description, looks for the nodes that connects to the failure switch according to the configuration of the target system, then determines the nodes that need to be injected a network interface failure, after that, multiple fault-injection commands are sent to the nodes simultaneously.

4 Experiment Results

4.1 Methodology

We evaluate the HPC-SFI in a cluster environment with four nodes. The experimental target node is an Intel CPU-based computer system running Linux operating system Ubuntu16.04 with 1 GB of memory and 4.13 kernel versions. Unlike current fault injection tools, our HPC-SFI inject hardware/software-failures in real physical systems. So application-level workloads do not affect HPC-SFI fault injection. For the experiment to clearly show the effectiveness of fault injection, we use matrix multiplication as the workload, which consists of multiple loop of the initialization step of input matrix data and the multiplication step.

In our experiments, firstly, we run the workload on the target system and start the HPC-SFI fault injection tool; the user then generates a fault of the specified type by configuring the relevant parameters; after that, the master node sends the message parsing package to the target node, and performs fault injection on the target node. After the fault-injection, the main control node waits for a specific time interval to observe the response of the target system, collect and analyze the fault response records. Specifically, we measure fault injection latency as well as the probabilistic distribution under pseudo-random mode. The fault-injection latency is the elapsed time from the sending of a fault/error injection command in the master to the completion of fault-injection in the specified node.

4.2 Experiment Results

In the experiment, we mainly test the probability of node failure injection and the coverage of the test between nodes. Tables 2 and 3 show the statistical analysis results of fault injection data. Each table provides specific information about the behaviors of the fault injection system. From the system perspective, HPC-SFI can effectively inject the faults to the system node. Further influence of the running state of the application, and diagnose the effectiveness of fault injection through the abnormal behaviors of the application. In the experiment, we realize three types of fault injection. At the same time, we also test the tool from the three directions: fault injection probability, node coverage range and fault delay.

Table 2. Results of injecting memory and processor faults on the single node.

Faulty component	Failure activation probability	Monitoring information	Times of fault node detected	Detected probability	Average fault-injection latency (ms)	Phenomenon
Memory (Injection times T = 20)	100%	System log	20	100%	163.67	The system log shows that the memory contents have been rewritten. The process interrupts an error and sets the SIGNAL
		Processor	19	95%		
	50%	System log	11	55%	226.67	
		Processor	11	55%		
Processor (Injection times T = 20)	100%	System status	20	100%	35.71	The process is forcibly stopped; or the node crashes and needs to be restarted
		Processor				

The date in Table 2 is based on the single-node fault injection, and is capable of verifying the validity of injecting memory and processor faults. To better simulate the random generation of node failure in a cluster. We set the trigger probability in the model-generation module. In the single-node memory fault injection experiment, we set the trigger probability to be 50% and 100% respectively. Through the fault injection of the tool, it can be found that the fault can be injected into the specified position accurately. By observing the system log and processor behavior, and testing the corresponding injection delay, the validity of the tool can be proved. In the process of the memory fault injection, different injection locations and injection time affect different system behaviors. Since the data required to execute the partial loader code has been loaded into the cache, subsequent data does not have to interact with memory, so the fault diagnosis is delayed and the fault is not fully reflected in the application process, but the system log file can reflect the effective injection of the fault.

Table 3. Results of injecting disk faults on the nodes in the cluster.

Faulty component	Number of fault nodes	Node	Selected times	Detected probability	Average fault-injection latency (ms)	Phenomenon
Disk (Injection times T = 50; Node = 4)	2	Node 1	35	48%	41.09	A partition that is not mounted cannot be mounted properly and displays a disk error; the partition being mounted cannot be read or written properly
		Node 2	39	50%	50.67	
		Node 3	41	46%	49.29	
		Node 4	35	56%	45.45	
	3	Node 1	24	70%	46.14	
		Node 2	25	78%	53.17	
		Node 3	23	82%	50.81	
		Node 4r	28	70%	49.17	

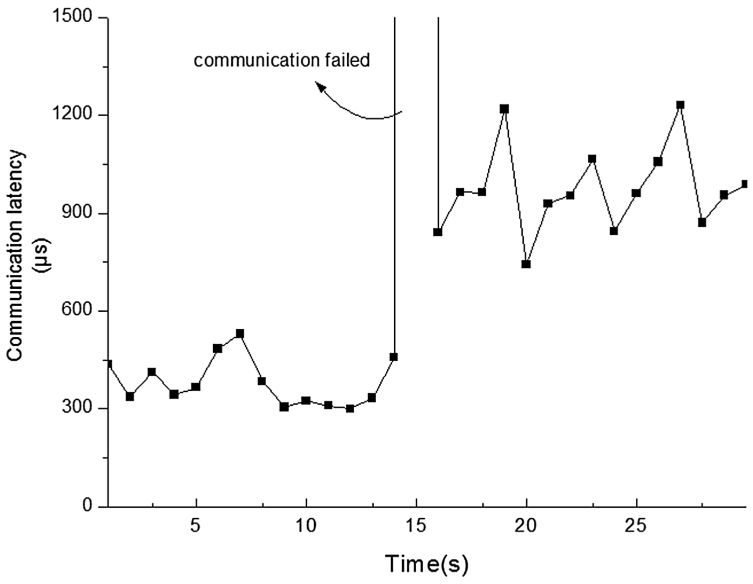

Fig. 2. Latency of the communication between nodes.

The failure model of the fault injection tool is generated by the user through the configuration description file. The tool defines the fault injection probability model by setting relevant parameters according to the specific fault injection requirements. According to the four-tuple of fault injection pseudo-random probability model, we set the test node range R to 4, the specified injection node number NUM is 2 and 3 respectively, and the F fault type is a disk fault. We test the fault injection for parallel file systems with the node coverage, and the results are shown in Table 3. Through the fault injection of the tool, it is proved that the setting of the node parameter can be applied to the fault model, and the fault injection probability conforms to the parameter setting. This kind of fault injection tool based on pseudo-random probability model can be more close to the actual system environment in the choice of fault injection.

We affect normal node communication through the fault injection tool, and the impact of the failure is reflected in the latency of node information interaction. In the HPC system, it is necessary for the fault injection tool to restore part of the fault and maintain communication, and at the same time, tries not to cause permanent failure to nodes to ensure the experiment. As showed in Fig. 2, when the fault is being injected, the message transmission delay between nodes changes dramatically and is much greater than the normal time. After a certain time, the fault will recover itself and normal communication between nodes will resume.

5 Related Work

Fault injection technique provides the capacity of evaluating the risibility of HPC system with synthetic failure occurring in hardware, system, as well as applications. To emulate the effect of failure in HPC, extensive studies have been conducted to explore different fault injection methods. Generally, current work fall into three categories: hardware-implemented fault injection, software implemented fault injection, and simulator and virtual machine based fault injection.

Hardware-implemented fault injection works by triggering errors in hardware with specialized device, such as setups producing electromagnetic interference and radiation [2, 3], or changing the voltage or current of target circuit board [4]. This type of method can mimic failure caused by environmental factors. However, it increases the risk of damage to the target hardware. Furthermore, it is difficult to control the fault location and triggering time.

Software-implemented fault injection method, on the other hand, generates emulated fault effect by inserting instruction into the application or triggering specific system command. Compared with the hardware-implemented counterpart, this type of method provides more flexibility and controllability. Therefore, many existed fault injection tools are implemented in software. For instance, Han et al. [5] proposed DOCTOR, a software implemented fault injection tool that injects hardware and software fault for distributed real-time system. Taking advantage of function available in the operation system, DOCTOR is able to inject architecture-independent hardware errors e.g., memory, CPU and communication fault and their combination, as well as system-level error. What's more, DOCTOR introduces temporal types and probability distribution for fault injection, which empowers the function of injecting realistic errors. Carreira et al. [6]. presented Xception, which injects realistic system and application faults in software by programming the debugging hardware available in modern processors. Based on the hardware feature, faults injected by Xception can affect any process running on the target system. Since software implemented fault injection dependent on the available function in target operation system and hardware, the variety of fault may be limited.

Simulator and virtual machine based fault injection, which emulates fault by revising the instruction of a virtual machine or serves as a module of full-system simulator, e.g., Gem5 [7], is eligible for performing various kinds of synthetic faults to HPC. Since this type of method is independent of hardware architecture and easy to control triggering time and location, there has become an increasing amount of literature focus on virtualization-based fault injection. For example, Guan et al. [8] proposed a fine-grained fault injector on top of QEMU [9] that emulates both software error and hardware failure by intercepting and corrupting instruction issued by an application before they be sent to the host kernel. Levy et al. [10] designed a virtualization based fault injection framework that mimics hardware errors both in individual node and across nodes in HPC system. By integrating error executor running on a virtual machine monitor in each node with error scheduler for dispatching deterministic and stochastic errors across HPC system, their framework is able to mimic more realistic faults in HPC. The main disadvantage of the simulator and virtual based fault injection method is the performance overhead, especially those work in full-system simulator.

6 Conclusion

In this paper, we propose a system level fault injection tool called HPC-SFI for HPC system. It utilizes software implemented fault injection to inject hardware/software failure into actual physical systems, and is intended for validation and evaluation of high-performance computing systems. We implemented a fault injection tool, HPC-SFI, which injected HPC systems with three types of failure: in-node faults, interconnection network failure, and storage/parallel file system failure. It can generate faults according to the parameters and probability, making it closer to the actual system. To avoid some irreversible damage to the node, it can be recovered after a specified time period. HPC-SFI was implemented on a linux cluster system, and extensive experiments were conducted, demonstrating its power and utility. We are also exploring the issues about the specification of fault injection and more extensive fault coverage. After these extensions, we will conduct more practical experiments.

Acknowledgments. The research presented in this paper has been supported by National Key R&D Program of China under grant No. 2016YFB0200100 and Natural Science Foundation of China under Grant No. 91530324.

References

1. The Top500 List, June 2018. http://www.top500.org
2. Karlsson, J., Liden, P., Dahlgren, P., et al.: Using heavy-ion radiation to validate fault-handling mechanisms. IEEE Micro **14**(1), 8–23 (1994)
3. Gunneflo, U., Karlsson, J., Torin, J.: Evaluation of error detection schemes using fault injection by heavy-ion radiation. In: 1989 The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers, pp. 340–347. IEEE (1989)

4. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault injection techniques and tools. Computer **30**(4), 75–82 (1997)
5. Han, S., Shin, K.G., Rosenberg, H.A.: Doctor: an integrated software fault injection environment for distributed real-time systems. In: 1995 Proceedings of International Computer Performance and Dependability Symposium, pp. 204–213. IEEE (1995)
6. Carreira, J., Madeira, H., Silva, J.G., et al.: Xception: software fault injection and monitoring in processor functional units (1995)
7. Binkert, N., et al.: The Gem5 simulator. SIGARCH Comput. Arch. News **39**(2), 1–7 (2011)
8. Guan, Q., Debardeleben, N., Blanchard, S., et al.: F-SEFI: a fine-grained soft error fault injection tool for profiling application vulnerability. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp. 1245–1254. IEEE (2014)
9. Bellard, F.: QEMU, a fast and portable dynamic translator. In: USENIX Annual Technical Conference, FREENIX Track, vol. 41, p. 46 (2005)
10. Levy, S., Dosanjh, M.G.F., Bridges, P.G., et al.: Using unreliable virtual hardware to inject errors in extreme-scale systems. In: Proceedings of the 3rd Workshop on Fault-tolerance for HPC at Extreme Scale, pp. 21–26. ACM (2013)