Computer Science and Information Systems 18(3):771–790 https://doi.org/10.2298/CSIS200620040W

End-to-End Diagnosis of Cloud Systems against Intermittent Faults

Chao Wang[1,3], Zhongchuan Fu[2,*], and Yanyan Huo[1]

[1] Computer School, Beijing Information Science and Technology University,
North 4th Ring Mid Road 35,
100101 Beijing, China
wangchao@bistu.edu.cn
[2] Computer Science & Technology Department, Harbin Institute of Technology,
Xidazhi Street 92,
150001 Heilongjiang, China
fuzhongchuan@hit.edu.cn
[3] Beijing Advanced Innovation Center for Materials Genome Engineering,
North 4th Ring Mid Road 35,
100101 Beijing, China

Abstract. The diagnosis of intermittent faults is challenging because of their random manifestation due to intricate mechanisms. Conventional diagnosis methods are no longer effective for these faults, especially for hierachical environment, such as cloud computing. This paper proposes a fault diagnosis method that can effectively identify and locate intermittent faults originating from (but not limited to) processors in the cloud computing environment. The method is end-to-end in that it does not rely on artificial feature extraction for applied scenarios, making it more generalizable than conventional neural network-based methods. It can be implemented with no additional fault detection mechanisms, and is realized by software with almost zero hardware cost. The proposed method shows a higher fault diagnosis accuracy than BP network, reaching 97.98% with low latency.

Keywords: cloud system, intermittent fault, fault diagnosis, end-to-end, LSTM, PNN.

1. Introduction

The diagnosis of intermittent faults has drawn increasing attention in recent years. This problem is challenging because of the random manifestation of such faults due to intricate mechanisms. This can be mainly attributed to two reasons: a) The long time operation, high-load operation, and large cluster scale could more easily lead to phenomena such as PVT variation, cross talk, and interference, as the computing density increases (along with energy throughput) in cloud systems; b) On the other hand, aggressive chip feature sizes increase the hardware fault susceptibility of the single device itself [1].

* Corresponding author

Hardware faults can be classified into 3 categories: transient, intermittent, and permanent faults, depending on the duration. Transient faults often manifest as bit-flips and were first detected through a radioactive material contained in the chip package. High-energy radioactive particles, such as thorium and uranium, in the package emit α particles with energy > 8 MeV. When the accumulated electric quantity exceeds the charge threshold, the behavior of the PN junction changes, resulting in a bit-flip [2]. The duration of this type of fault is in the picosecond scale; it can be recovered by writing or refreshing. By contrast, a permanent fault is due to the aging of components or irreversible physical damage such as open or short failures in the circuits [3]. Permanent faults need to be replaced or repaired. Existing state-of-the-art diagnosis technologies are mainly designed for transient and permanent faults.

The mechanism of intermittent faults is complex, and was put forward as early as the 1970s [4]. The causes of device failures include time-dependent dielectric breakdown (TDDB), negative bias temperature instability (NBTI), electromigration (EM), stress migration (SM), and thermal cycling (TC) [5]. An intermittent fault is non-periodic, i.e., the time, frequency, probability, and amplitude of fault occurrence are random [6]. As reported, faults that occur in electronic devices are typically intermittent. Intermittent faults in integrated circuits are 10~30 times more frequent than permanent faults [7]. An error report [8] from Microsoft Windows on 950,000 personal computers showed that approximately 39% of the hardware errors reported in microprocessors are intermittent faults.

Most existing fault diagnosis technologies are based on replay, i.e., after detecting the fault, the process instance is executed again on the standby core, and the former and latter are then compared for the diagnosis. This type of method is only applicable to distinct transient and permanent faults, because intermittent faults occur non-periodically and are not necessarily reproduced in the process of replay. This let us to conclude that the diagnosis of intermittent faults is challenging, from the embed devices to the cloud computing systems (with redundant threads or cores, but not avail against the uncertainty and propagation thereafter). Raghavan [9] compared the outputs of a tested circuit and reference circuit, and distinguished permanent and intermittent faults based on whether the number of faults exceeded a certain threshold. The number and threshold of faults are typically determined with respect to conditions such as the fault rate. To diagnose intermittent faults, Lafortune adopted the monitoring theory to study the diagnosability of discrete event systems (DESs), to check whether the fault can be diagnosed within a limited time [10, 11]. Due to the assumption that the fault type is known (assuming that the known fault is intermittent), the purpose is to identify how the system works in the recovery state against an intermittent fault, but not to diagnose an unknown fault, based on observable events, to be an intermittent fault or not. Therefore, the diagnosability of intermittent fault cannot be analyzed.

A novel intermittent fault diagnosis algorithm for cloud systems is presented to overcome the above limitations. Our contributions are as follows: (1) End-to-End. To avoid the heavy reliance on environment feature extraction, this method is intuitively designed as an end-to-end diagnosis method, which, although requires information selection, does not use any potential function; it explores the best characteristic representation to solve problems from the perspective of "intuition"; (2) Covers all the hardware types. Unlike most conventional methods, this method covers all the hardware types and the fault locating responsibility, meaning that this method needs to identify

each instance to be a golden run or an isolating faulty category (transient/ intermittent/ permanent faults), and also to locate where the fault originates from; (3) No additional fault detection mechanism. This method does not rely on additional fault detection mechanisms. It uses only hardware interrupt handlers as the basis for inspection; these are commonly used in central processor units, and it is realized via software, so the hardware cost is almost zero. The experimental results show that the diagnosis accuracy reaches 97.98% for all the three types of faults (transient, intermittent, and permanent faults).

The rest of this paper is organized as follows. Section 2 presents the related work in literature. Section 3 describes a novel end-to-end diagnosis framework, including corresponding algorithms. In Section 4, the method is validated with experimental work. Section 5 concludes the paper.

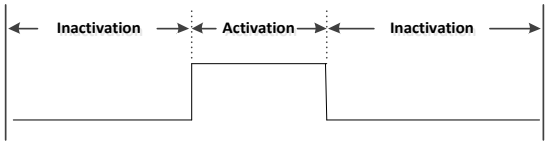
2. Related Work

We would like to present the research work in literature of the fault diagnosis area, especially in the hardwired faults those originated in the computer devices. In this part of work, the description of transient, intermittent, and permanent fault models, the fault diagnosis method and fault injection technologies are introduced.

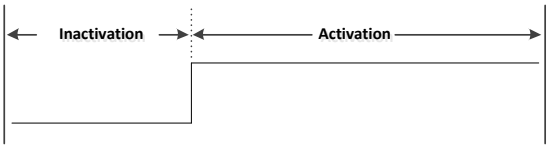
2.1. Fault Models

The pulse description method can uniformly describe the hardware fault models, by using the activation time and the inactivity time as the parameters during the fault occurs. In the case of irradiation, when the illuminated high-energy particles reach the fault threshold, the transient fault will be triggered, causing a bit flip. As the fault duration increases, the energy is released and the transient fault disappears(see Fig. 1(a)). A permanent fault is an irreversible physical defects in the circuit, and a fault phenomenon will always exist (see Fig. 1(b)).

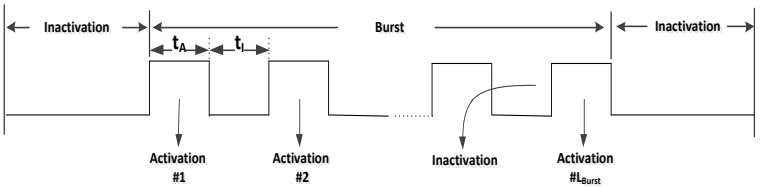
The intermittent faults are different from these two. The occurrence and disappearance of intermittent faults happens mutually (see Fig.1(c)). The fault location is fixed (same as transient and permanent faults). In fact, the intermittent fault model with 101 order duration of the clock cycle has now been accepted by the academic community, and thus, is adopted in this paper and is applied at different levels such as processor structure, virtual machine monitor, operating system and even application level.



(a) Transient fault model



(b) Permanent fault model



(c) Intermittent fault model

Fig. 1. Pulse-based description method for hardware faults [12, 33].

2.2. Fault diagnosis methods

Research that designs scheme for the post-silicon debugging mechanism records the footprint of every instruction as it is executed in the processor [13-15]. Some of them (e.g., IFRA[16]) requires the presence of hardware-based fault detectors to limit the error propagation, while others are implemented in a hybrid hardware-software manner, and with no additional detectors [17, 18]. Carratero et al. [19] propose their method to diagnose faults in the load-store unit (LSU) which is performed during post-silicon validation, and it only covers design faults. In contrast, SCRIBE [20] is proposed to diagnose intermittent faults during regular operation. After the fault is detected, the program is replayed on the standby core, and a data dependence graph (DDG) is constructed by extracting the runtime information (microstructure-level devices). By comparing the data flow graphs of two runs [21], the diagnosis and location of the intermittent fault are realized. Our work is similar to theirs in some aspects. However, as SCRIBE's potential assumption that the fault type is known (assuming that the known fault is intermittent or permanent), the purpose is to diagnose how the system is currently in a recovery or intermittent fault state based on observable events. Therefore, in fact, the diagnosability of intermittent fault are remained unsolved, and additional detection mechanism is still needed by this method.

Hari et al. designed a trace-based fault diagnosis (TBFD) mechanism to diagnose permanent faults. Although the diagnosis accuracy reached 95%, heavy-weight overheads, such as hardware buffers and re-executions, were required [22]. Furthermore, TBFD is only effective for permanent faults. Considering the burst and non-periodic characteristics of intermittent faults, TBFD is not an alternative solution for intermittent fault diagnosis. Deng et al. proposed a stochastic automata-based method that can diagnose both of the permanent fault and the intermittent fault. They set up a finite automaton model by introducing the fault identification mechanism, wherein the state transformation of the system is invested, and the probability of the fault event is made out [23].

The above methods depend on the scale of sample space: few samples cannot guarantee the accuracy of the diagnosis, which in turn can easily cause false alarms. As the existing samples are often limited in the real-world [24], fault injection is an effective method to accumulate the fault instances.

2.3. Fault injectors

Fault injectors are developed and realized toward upper levels in view of systematization. VFIT [25], INJECT [26], and VERIFY [27] are fault injection platforms developed on very high-speed hardware description language (VHDL), supporting fault models on the switch-level, gate level, and register transfer level (RTL). Wang et al. extended their fault injection simulator to multi-core architecture. They selected the UltraSPARC processor (8 cores, 64 threads) as Device Under Test (DUT) to characterize the effects of intermittent faults at the RTL level, and showed that some systematic events can be used as detection symptoms [28-29]. Rashid set up a pure software-based fault injector that is designed on SimpleScalar, and investigated the characteristic of intermittent faults at the application program level (Spec CPU2006) [30]. Hu et al. set up a system-level fault injection platform based on the Simics simulator, and studied the impact of hardware fault on a multi-core system through software simulation, including operating system and application program [31]. Le and Tamir proposed fault injection tools based on cloud environments, taking advantage of virtualization environment (virtual machine monitor) to implement a fault injection interface toward the upper layers [32]. As fault injection modules are (and can only be) implemented in a virtual machine monitor, only misbehaviors of the guest operating system fall into the observation scope and can be tracked.

In this study, the cloud platform is selected as the injection target. Unlike the above fault injector, this work is not implemented merely “on” the cloud (the fault behavior propagation path only covers the operating system level and above); in fact, this work is different in that the virtualization firmware can be tracked even at the CPU structural level, which is beyond the operating system level. Thus, the fault propagation behavior can be tracked with more accuracy than injectors set up on the cloud.

3. Approach

This paper presents an end-to-end fault diagnosis method. The fault log is recorded in the fault injection camp in the cloud environment. Based on the system level run-time information, features are automatically extracted and inputted to the neural network. This method covers all the hardware types as the target fault set, including transient, intermittent, and permanent faults.

We first perform fault diagnosis based on a BP neural network through the statistical analysis of the log. Although artificial feature extraction is less computationally complex than the end-to-end method, there are drawbacks in the way it relies on manual feature extraction, which has two disadvantages: First, the selection of the features needs to be conducive to the classification. Therefore, features are combined through statistical or potential function methods for processing. This method strongly depends on the quality of the feature extraction, even more important than the learning algorithm used. For example, if the color of hair is extracted as a feature, the classification effect for gender will be poor regardless of the classification algorithm used. Therefore, features need enough training for design, which is increasingly difficult in the case of large amounts of data and complex systems. In addition, useful information may be potentially lost in the calculation of the original features. Second, the data element in the feature set may change (information or attributes need to be updated) depending on the operating environment in order to avoid the lack of generalization ability, and the repeated tuning and optimization processes for evaluating how the extracted features may influence the back-end performance, which may increase the time cost of model development. Therefore, an end-to-end diagnosis framework for system-level symptoms is proposed in this paper, providing an efficient solution to the implementation of intermittent fault diagnosis.

3.1. Challenges and solutions of end-to-end model

In the non-end-to-end algorithm, a significant amount of preparatory work is required. For example, in speech recognition, "phoneme" has been invented by linguists. Although it improves the efficiency in the processing step, it will undoubtedly lead to other information loss in the speech. The algorithm requires less data. However, the feature extraction depends on humans, and the feature needs to be redefined for application scenario migration (such as changing language), so the generalization ability is not high.

Hence, the end-to-end method has been proposed, in which the original data are pre-processed and selected as features that are learned without any potential functions. Hence, it can be integrated into the algorithm without human intervention, in order to explore the best characteristic representation to solve problems from the perspective of "intuition". As a result, the input (original data or feature sequence) and the output (fault categories or locations) have been directly connected to both ends of a neural network. However, the end-to-end learning algorithm does not require much human intervention, but it needs a lot of labeled data.

Based on a fault behavior tracking (FBT) system [33], we have applied a two-month period fault injection campus to obtain the systematic-level fault propagation behavior in the cloud computing environment. We obtained statistics from 42,000 experiments on fault injection under SPEC2006 workloads, including eon, gcc, parser, perlbmk, and twolf. For each instance, one of the three types of faults is chosen and injected into the target fault location. We set a time window (within the time of 1,000,000 instructions starting from fault injection) and collected the system-level fault propagation behavior sequence generated in this window. For intermittent faults, an total of 24,000 runs (300 injections * 4 units * 5 benchmarks * 4 Lburst) were conducted; for transient and permanent faults, we conducted 12,000 and 6,000 runs, respectively, since there are two types of permanent faults, namely permanent stuck@0 and permanent stuck@1, compared with the transient faults, which are only of one type. Based on this behavior, the input neural network extracts the features and carries out fault diagnosis. Currently, the simulator covers all the hardware types as the target fault set, including transient, intermittent, and permanent faults, and supports fault injections into four targets, namely the Address generator, Decoder, ALU_FPU, and Register Files in the processor, and monitors the run-time log trace from the instruction buffer and state registers. We developed FBT modules to monitor the software stack.

Given that millions of experimental instances are required to produce numerical labeled data for training the end-to-end framework, we implemented fault injection automatically in the FBT, wherein blue screen recognition and dead loop detection were developed in the controller module, to recognize system crashes due to illegal memory address access, trap stack overflow, and/or other severe perturbations.

3.2. Overall architecture

The reliability modules include the fault injector, fault tracer, and analyzer modules. In **Step 1**, we developed the fault injector module in the FBT to inject the three types of faults (transient/intermittent/permanent) into the specified location in the target unit. The target system is a multi-layer cloud system simulator, wherein the CPU/memory/hard disk is located beyond the VMM and guest operating systems. We adopted the prototype of UltraSPARC T2 processor as the target CPU. UltraSPARC T2 is a commercial chip multi-threading (CMT) processor, which has eight 64-bit cores and 8/16 threads in each core. Instead of exploiting instruction-level parallelism (ILP) and deep pipelining, this processor model achieves a good performance by taking advantage of thread-level parallelism (TLP), which is an optimized CPU model for cloud computing environment, instead of using the ILP architecture.

The cloud software stack, comprising a VMM layer and the operating system for control domain and other virtual domains, is overlaid on top of the simulated hardware. Inside these domains, user applications (in our fault injection campaigns, the benchmark) are processed. The execution environment includes the computer hardware and host operating system. The latter is responsible for the simulator and other fault injection relevant modules. Below the host operating system is a (real world) hardware computing device that is responsible for executing all the software layers in **Step 2**, and the logs are then recorded in the host operating system in **Step 3**. The system-level symptoms are collected so that the fault propagation can be logged at all levels.

Step 4: Feature selection

When using the machine learning technique, feature selection is the most important part. Based on the statistical distribution we just proved, we can take the number of times of a system call shown up in the trace as the major feature and other features, such as the trap level and high OS, as the complement features (unlike feature extraction, feature selection does not require a calculation process for the potential function, which belongs to the original data, because we cannot and do not need to input all the original data into the neural network). The exceptions and interrupts in the cloud environment are collectively referred to as trap. In SPARC architecture, the related attribute values of the trap are stored in specific registers (as listed in Table 1). TL is the trap-level register, which specifies the trap nesting level of the current program state. Under normal circumstances, the value of TL is 0, which means no trap. When the processor enters a trap, the value of TL is increased by 1. When the nesting level of the trap is greater than 1, nest failure occurs. The SPARC architecture requires that at least five layers of nesting are supported. A nest fault is determined by the value of TL. When TL is greater than or equal to 2, nest fault occurs. TT is the trap-type register, indicating the trap-type number. The values of CCR, ASI, pstate, and CWP are also saved in the TSTATE register. The HP and P states represent the privilege level of the processor, indicating hypervisor authorization and operating system administrator, respectively. When a trap occurs, the hardware will automatically save PC/NPC to TPC/TNPC, and save CCR/ASI/pstate/CWP to TSTATE. Otherwise, the trap state program counter (TPC), trap state next program counter (TNPC), and TSTATE are saved in the hardware register stack. The CPU then enters the privilege execution mode and jumps to the trap vector entry to execute the relevant trap service program.

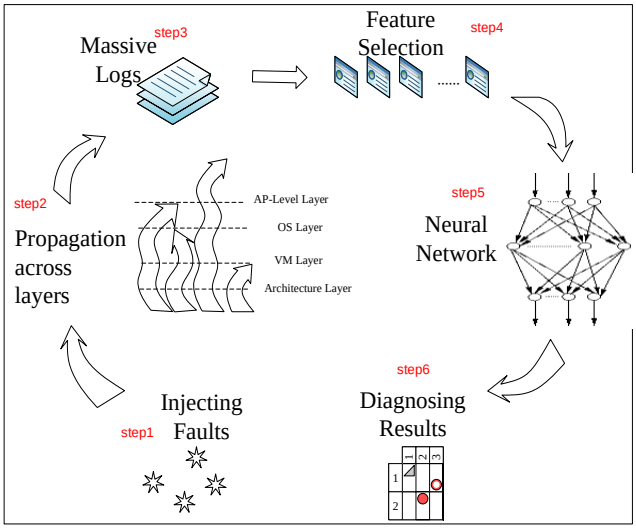


Fig. 2. Block diagram of the proposed end-to-end diagnosis algorithm

High OS: the trap handler only takes a small piece of the coding fragment, except in two cases: 1) to allocate time slices to the application, the operating system may take a longer time to execute. And we record that the maximum continuous instructions is 10000, by tracking the instructions running in the priviledged mode (operating system); 2) to execute the system call procedures, the operating system executes 105 or 106 continuous instructions before returning to the unpriviledged mode (application program). Therefore, under normal states, the number of continuous instructions executed in the priviledged mode will not exceed 106. When this threshold is exceeded, the behavior is considered abnormal.

Table 1. Functional trap registers.

Register	Description
TL	Register to record Trap Level
TT	Register to record Trap Type
TSTATE	Register to record Trap State

Steps 5 & 6: Diagnose algorithms

In the process of fault diagnosis, both of the two learning strategies have been investigated--offline and online. By analyzing the fault behavior (based on the log files), it is not difficult to find that the sample can be regarded as a sequence. For each fault injection simulation instance, several trap events are generated and then logged. Based on this, a sequence can be simply setup as sample towards a learning strategy. In this paper, the method based on the long and short term neural network is adopted, that is, the trap sequence is constructed as the input vector to input to the long short term

memory (LSTM). Before the diagnosis framework starts to works, it requires to collect the entire trap event as the input sequence (from the beginning of the simulation to the finish), so it is called the offline learning strategy; on the other hand, each fault can be treated as an event that needs to be diagnosed immediately, and hence the serialized data can be expanded into vector data and submitted one by one. This are often called the online learning strategy. We implement the online mode based on Back Propagation Neural Network (BP) and Probabilistic Neural Network (PNN), respectively. The performance of the learning strategies will be discussed in section 4.

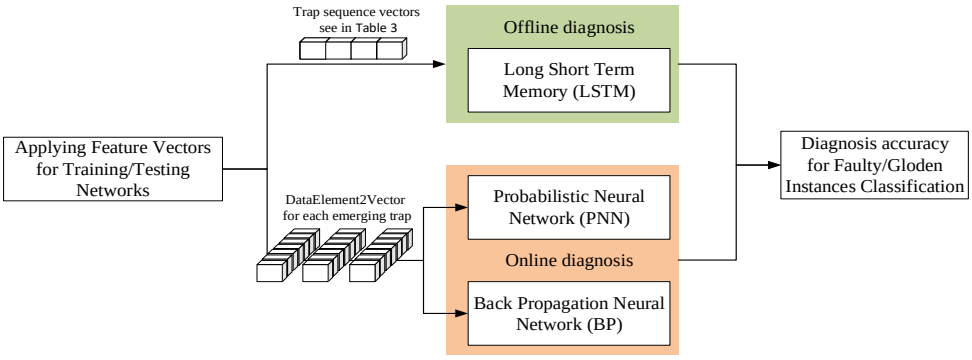


Fig. 3. The diagnosis framework consists of offline and online learning strategies.

Offline learning strategy

LSTM. In the course of training, RNN neural network often has gradient disappearing or exploding, so Hochreiter et al. [34] put forward long short term memory neural network. This problem is well overcome in LSTM by adding three gate structures: forget gate, input gate and output gate, to keep and update the status information of each unit module. The input gate receives the current information of the system; the forgetting gate filters the information and discards the useless memory; the output gate filters the value of the next hidden state. In this scheme, the output result is defined as the fault categories, in which we can select the maximum value as the diagnosis result. Cross entropy loss is chosen as the loss function, which is suitable for multiple classifiers. There are two parameters for cross entropy: input value and label, representing the specific gravity of classification of the samples and the category index [0, n-1]. In Equation 1, where is the true value and is the predicted value.

$$\mathrm{loss} = -\sum_{k=1}^{N} (m_k * \log n_k) \tag{1}$$

Online learning strategy

BP neural network. This is an artificial neural network based on the learning mechanism of back propagation. In the BP neural network, linear transformation is used to map nodes in the input layer to nodes in the hidden layer. The activation function of

hidden layer and the linear transformation are co-operated to map nodes from the hidden layer to the output layer. The hidden layer can be one or more layers. We adopt softmax to be the activation function, which converts each vector value to the [0, 1]. See the calculation formula in Equation 2:

$$f_i(x) = \frac{e^{(x_i-\alpha)}}{\sum^{j} e^{(x_j-\alpha)}}, \alpha = \max(x_i) \quad (2)$$

Wherein x_i is the i_{th} element in the input vector, α is the maximum element among x_i.

PNN. Unlike BP network, probabilistic neural network is a forward propagation classifier that uses Bayesian decision theory to classify samples. Bayesian decision-making refers to taking the test sample as the classification with the highest probability. The PNN consists of four layers: one input layer, one output layer, and two hidden layers. The two hidden layers are the sample and competition layers. The neuron activation function of the sample layer is used to calculate the distance between the input value and the category center. If the distance is close to a center, the probability of this value in the corresponding area is set high. Theoretically, the output function of the PNN adopts the Bayesian classification method, wherein using Gauss function (equation 3) to compute the distance between input vector and center point in order to classify the data with the maximum probability.

$$\mathrm{y}_g(x;\sigma) = \frac{1}{l_g(2\pi)^{n/2}\sigma^n} \sum_{i=1}^{l_g} \exp\left(-\sum_{j=1}^{n} \frac{\left(x_{ij}^{(g)} - x_j\right)^2}{2\sigma^2} \right) \quad (3)$$

wherein n represents number of feature dimension, l_g represents the number of samples in the g_{th} category, x_{ij} represents the j_{th} data of the i_{th} neuron, and σ is a hyper parameter.

3.3. Implementation

The following assumptions about the system are illustrated before we introduce the working flow: a) we assume a commodity multi-core system in which all cores are homogeneous, and are able to communicate with each other through a shared address space. b) We assume the availability of a fault-free core to perform the diagnosis. This is similar to the assumption made by and Li et al. [34]. The fault-free core is only needed during diagnosis. c) Trap logic unit (TLU) in processor is hardened in need to assure the correct exception information is logged. Note that UltraSparc T2 processor provides two trap return instructions, retry and done. Retry makes trap return to the instruction where trap is raised, and re-executes the instruction again when the done instruction returns to continue with the program. When the system detects a fault, it may use the retry instruction to return to the abnormal instruction for re-

execution, or use the done instruction to transfer the trap to the operating system when the hypervisor may not be able to process the trap.

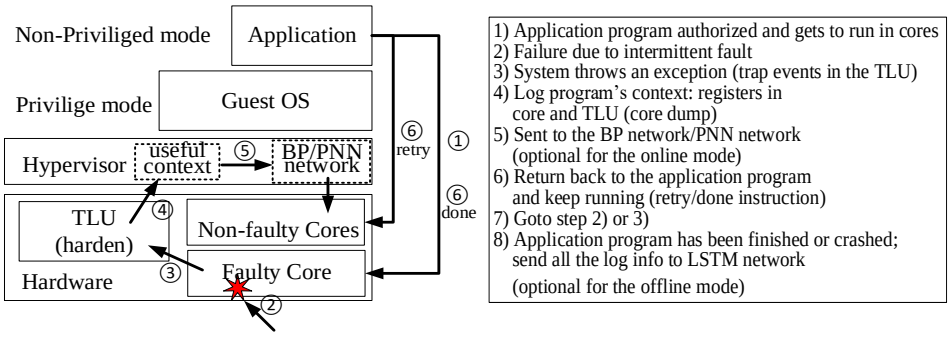


Fig. 4. Working flow of online and offline diagnose methods. The steps in the figure are explained in the box.

Overheads. Compared to other diagnosis schemes, our technique incurs low performance and power overheads with reasons as follows: a) as it initiates diagnosis only when error detection occurs, the diagnosis overhead is not incurred during fault-free execution; b) our scheme do not need to log the context information in the processor continuously (only when an error detection occurs, and not like SCRIBE [19] which needs to do this continuously); c) the complex task of figuring out the fault type and faulty component is done in software. Hence, the power overhead is low.

4. Experimental result

In this section, we evaluated the performance of the proposed end-to-end diagnosis framework against hardware faults for cloud computing systems. We used the FBT simulator based on the software asset management (SAM) to emulate the considered case studies.

Figure 5 shows the coverage of systematic-level fault behavior in the cloud system environment in our FBT simulator. high OS is large. In the transient fault, the coverage of high OS is the highest, in the permanent fault model, the coverage of high OS is the lowest, and almost 0 in the ALU and the decoder. The coverage of high OS decreases with the increase of the burst length. In the ALU, the coverage rate of high OS is significantly higher than that of other components. The overall coverage of nest is also high, and with the increase of the burst length, the coverage is significantly increased. In the transient fault model, the coverage is basically 0, which can be used as the diagnostic feature of the model.

In these traps, the coverage of 0x10 and 0x34 is high. In the ALU, the trap is mainly 0x34, which is caused by the address reading error of the ALU. In the decoder, the coverage of 0x10 and 0x34 is about 50%, which may be caused by the illegal instruction caused by bit flipping, or by the error of the target address or register number caused by the fault, resulting in the wrong instruction address, etc. In the program counter (PC)

register, the coverage of 0x10 and 0x34 is high, which may be caused by the change of PC value. The coverage of 0x10 is almost 0 in ALU, but higher in other components, which can be used as the diagnosis feature of ALU. 0x30 only appears in the faulty ALU, and 0xd only appears in the faulty PC register, which can also be used as feature of faulty location diagnosis.

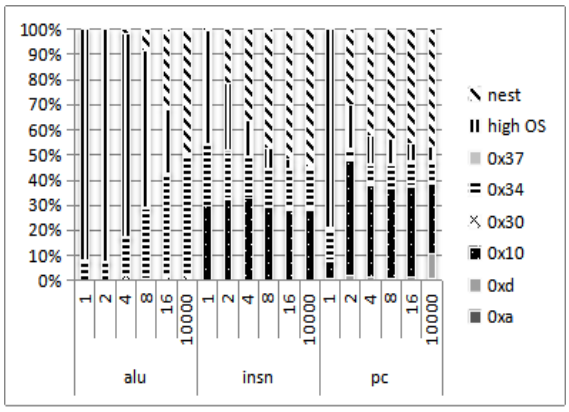


Fig. 5. Systematic-level fault behavior occupation

Offline diagnosis scheme(LSTM):

Cross entropy loss adopts “one hot” mode. As shown in table 2, when hidden reaches 249, the accuracy is the highest; when hidden is determined as 249, it is found that when batch is 50, the accuracy is the highest. With the Adam optimizer, the learning rate is 1e-4, the regularization parameter is 1e-5, the number of neurons in the hidden layer is 249, the batch is set to 50, and the accuracy is 59.8%.

Table 2. Hidden nodes and batch size tuning of LSTM.

Hidden nodes tuning				Batch size tuning	
Hidden	Accuracy	Hidden	Accuracy	Batch	Accuracy
25	53.40%	248	58.70%	16	55.50%
50	59.10%	**249**	**59.80%**	**50**	**59.80%**
100	53.40%	250	59.10%	100	55.50%
150	57.70%	255	58.40%	150	59.10%
200	56.20%	260	56.60%	200	54.50%
245	57.30%	300	58.00%	250	50%

Online diagnosis scheme(BP/PNN):

For BP network, we adopt Adam optimizer, the learning rate is 1e-2, the number of neurons in hidden layer is 150. The sample proportion of fault type is 334:1205:294, and the accuracy rate is 77.83%. After further learning with smote data enhancement strategy, the accuracy of the fault type (T/I/P) is 89.71%, and the loss is stable at about 0.4. Figure 6 (d, e, f) shows the difference between the real value of the fault duration and the diagnosis value after the data enhancement. The sample proportion of fault parts is 284:884:666; the accuracy of fault location is 82.30%, and the loss is stable at about 0.4. See Table 3 column "fault location" for the accuracy of each fault location. See Figure 6 (a, b, c) for the true value and diagnosis value of fault diagnosis.

For PNN network, the transmission factor is set to 0.007 after tuning, and the best accuracy is 97.98%. Figure 7 shows the difference between the test result and the real value (the yellow line represents the real value and the blue line represents the predicted value). It can be seen that compared with the BP network, the diagnose performance of PNN is much more stable.

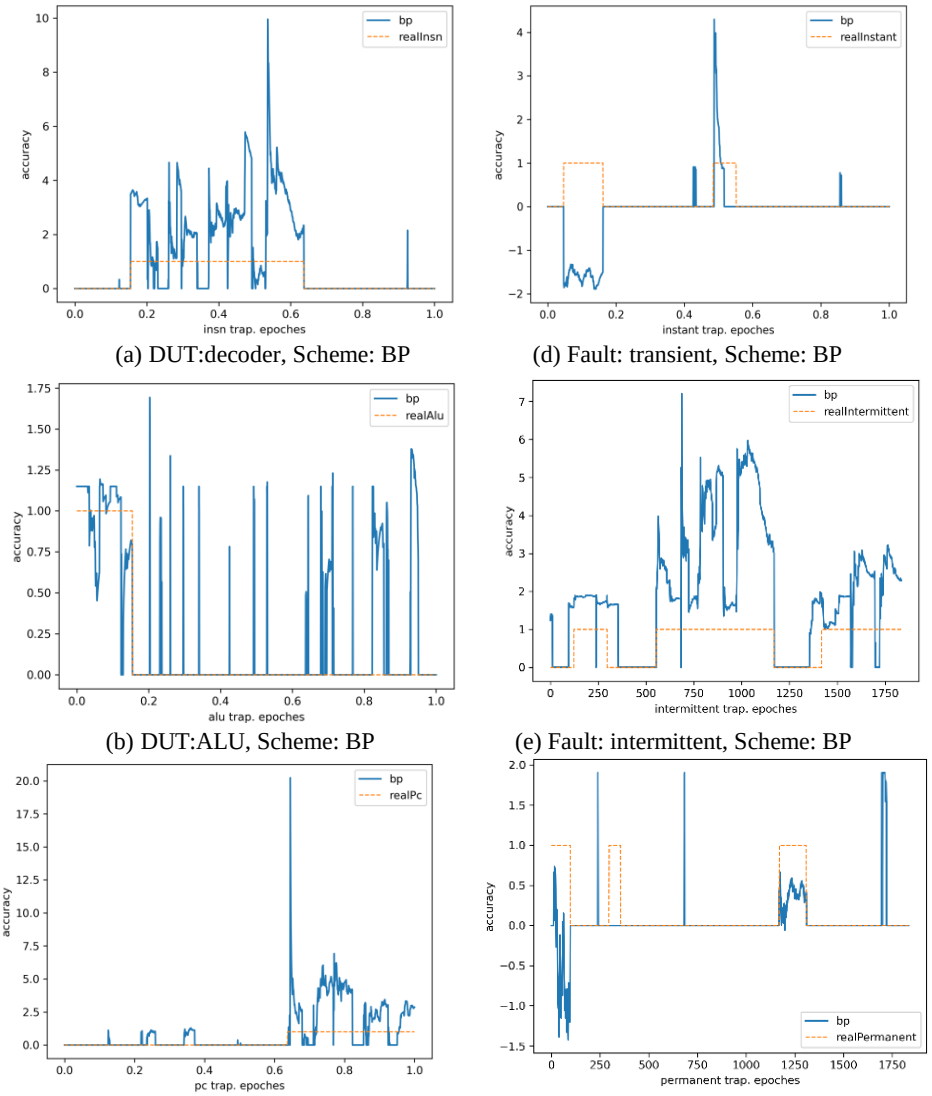
Figure 8 shows the training process of the LSTM network, in which the upper coordinate system represents the change of accuracy during training, the red line represents the change of test data loss value, and the blue line represents the change of training data loss value. The accuracy and loss changes of the BP network are shown in Figure 9. From the comparison, we can see that the BP network training process is stable, but the LSTM network training is more tough, in which the loss value and accuracy are changing unsteadily; then it can be concluded that the LSTM network is relatively poor in the ability to acquire knowledge from fault data compared with BP network.

Table 3. Diagnose accuracy of BP network.

Fault type (after SMOTE)			Fault location		
Fault type	Accuracy	Sample	DUT	Accuracy	Sample
transient	72.84%	334	decoder	75.09%	284
intermittent	97.68%	1205	ALU	84.18%	884
permanent	76.27%	294	PC	82.88%	666
total	**89.71%**		total	**82.30%**	

Table 4. Tuning of transmission factor in PNN network.

Factor	Accuracy
0.1	57.79%
0.05	72.06%
0.01	94.13%
0.008	95.64%
0.007	**97.98%**
0.006	**97.98%**



(a) DUT:decoder, Scheme: BP

(d) Fault: transient, Scheme: BP

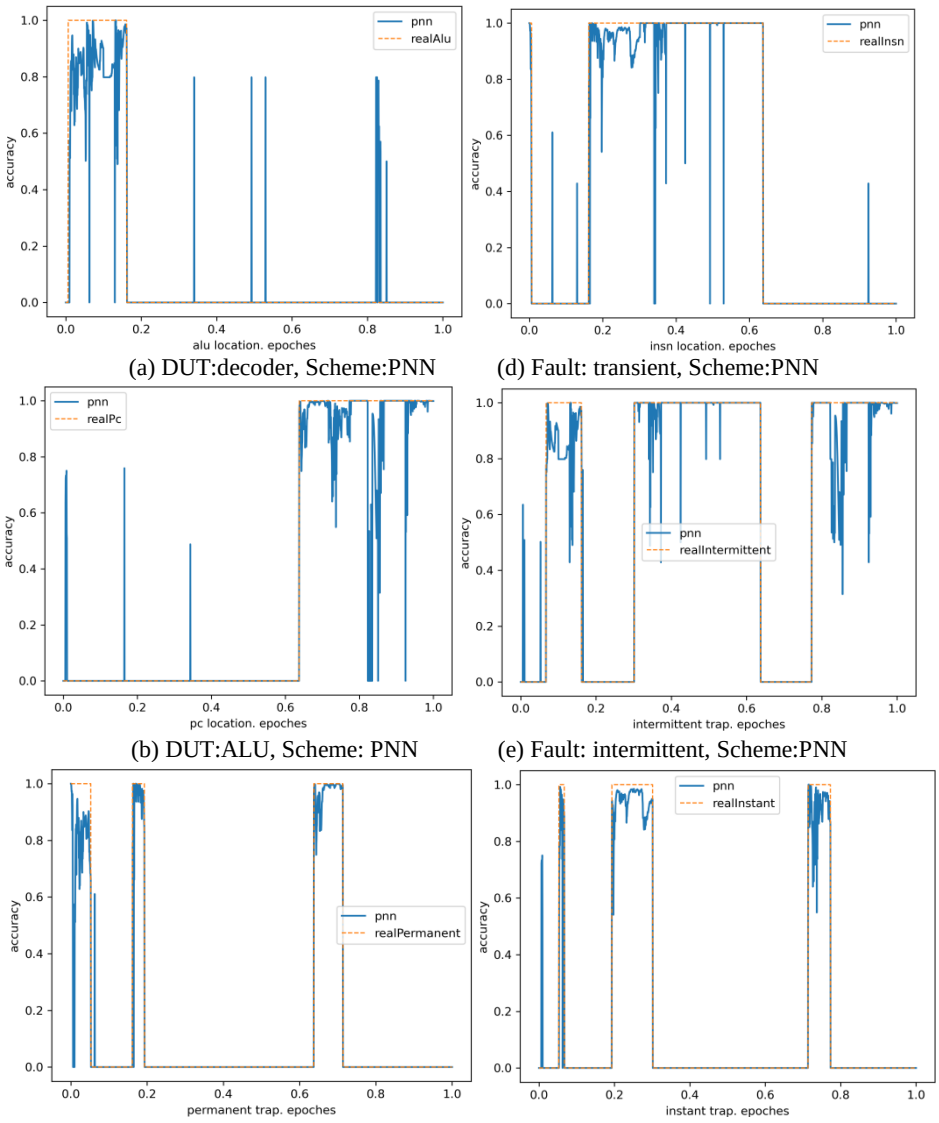
(b) DUT:ALU, Scheme: BP

(e) Fault: intermittent, Scheme: BP

(c) DUT: program counter, Scheme: BP

(f) Fault: permanent, Scheme: BP

Fig. 6. Diagnose result diagrams of BP network.



(a) DUT:decoder, Scheme:PNN (d) Fault: transient, Scheme:PNN

(b) DUT:ALU, Scheme: PNN (e) Fault: intermittent, Scheme:PNN

(c) DUT: program counter, Scheme: PNN (f) Fault: permanent, Scheme:PNN

Fig. 7. Diagnose result diagrams of PNN network.

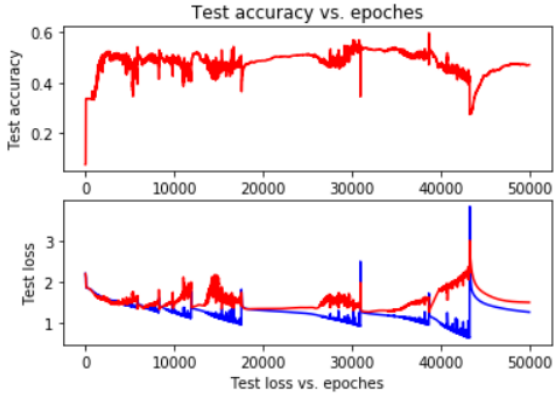


Fig.8. Test accuracy and loss of LSTM.

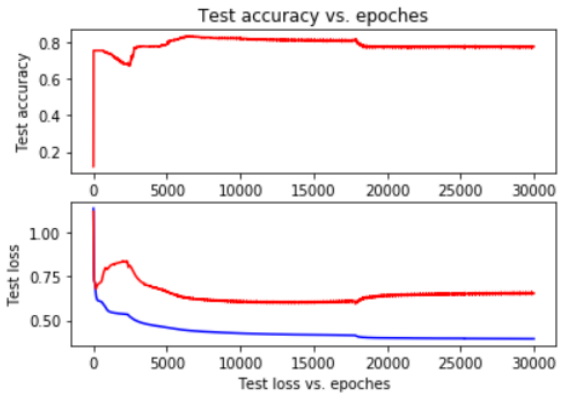


Fig.9. Test accuracy and loss of BP network.

The training process of the BP network is error back-propagation learning, and the basic requirement is that the error function has continuity (because it needs to ensure that the error function can be biased). Thus, the final fitting result of the BP network is a continuous function in multi-dimensional space; however, the general result of fault diagnosis is discontinuous: it is neither 0 nor 1, so the BP network has a larger error than the PNN network. In contrast, to classify data with the lowest risk, the PNN directly uses the Bayesian classification method based on the Gaussian density function. Hence, its output is either 0 or 1, so it has a higher fault diagnosis rate than the BP network.

The latency is 0.0952 seconds (BP) and 0.0286 seconds (PNN) alternatively for the online mode and 0.421 seconds (LSTM) for the offline mode. These algorithms run on Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, with 2 x 32 KB L1 cache and 2 x 256 KB L2 cache. The statistics do not include time for core dump. However, we observe that a considerable proportion of the system call traces of faulty instances is repeated in most of the cases, so there should be observable reduction in the latency. Accordingly, a software recovery mechanism is favorable. In addition, training of the network is done

offline. Hence, there is no need to recompute the weights of each neural connection when performing the diagnosis, thus saving significant time.

5. Conclusion

In this paper, we propose an offline/online diagnosis mechanism for cloud system against intermittent faults. We take systematic-level behavior as a high-level representation of fault behavior. We implement an end-to-end neural network-based method that takes advantage of the log information to perform feature selection. Then, we set up a unified diagnosis framework based on LSTM/BP/PNN classifiers. Among the three classifiers, the PNN performs best in diagnosis accuracy. It employs the Bayesian probability analysis method to make fault category and fault location close to the actual label. The offline training/online diagnosis ensures that this method can be implemented in firmware, with zero hardware costs.

References

1. Gil, P., Arlat, J., Madeira, H. et al, Fault Representativeness. Deliverable ETIE2. DBench European Project (IST-2000-25425).
2. Nishant J. George, Carl R. Elks, Barry W. Johnson and John Lach. Transient fault models and AVF estimation revisited//In International Conference on Dependable Systems and Networks (DSN), Chicago, IL, 2010:477-486.
3. Process Integration, Devices and Structures, The International Technology Roadmap for Semiconductors, 2012.
4. C. Constantinescu. Impact of deep submicron technology on dependability of VLSI circuits. Proceedings of Dependable Systems and Networks Conference, 2002:205-209.
5. Gracia-Moran J, Gil-Tomas D, Saiz-Adalid L J, et al. Experimental validation of a fault tolerant microcomputer system against intermittent faults[C]// IEEE/IFIP International Conference on Dependable Systems & Networks. 2010.
6. Tharf M S H. Computer modeling of electromagnetic interference, radiation, and crosstalk in electronic systems[M]. 1993.
7. C. Constantinescu. Intermittent Faults in VLSI Circuits[C]. Proceedings of the IEEE Workshop on Silicon Errors in Logic - System Effects, 2007.
8. Rashid L, Pattabiraman K and Gopalakrishnan S. Intermittent hardware errors recovery: modeling and evaluation. In: 2012 ninth international conference on quantitative evaluation of systems (QEST), London, 17– 20 September 2012. New York: IEEE.
9. Raghavan V. On Asymmetric Invalidation with Partial Test[J]. IEEE Transactions on Computers, 1993, 42(6): 764-768.
10. Lafortune S, Sengupta R, Sampath M, et al. Failure Diagnosis of Dynamic Systems an Approach Based on Discrete Event Systems[C]. Proc of the American Control Conference, Arlington, USA, 2001: 2058-2071.
11. Deng Guanqian, Qiu Jing, Li Zhi, Yan Ning. A Survey on Intermittent Fault Diagnosis Technology[J].Ordnance Industry Automation, 2015-01, 34(1): 15-20.
12. Daniel Gil-Tomás, Joaquín Gracia-Morán, J.-Carlos Baraza-Calvo, Luis-J. Saiz-Adalid, and Pedro-J. Gil-Vicente. Studying the effects of intermittent faults on a microcontroller[J]. Elsevier Microelectronics Reliability, 2012, 11(52):2837-2846

13. Rohan Garg, Tirthak Patel, Gene Cooperman, Devesh Tiwari. Shiraz: Exploiting System Reliability and Application Resilience Characteristics to Improve Large Scale System Throughput [C]// IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE, 2018, pp. 83-94.
14. Guanpeng Li, Karthik Pattabiraman. Modeling Input-Dependent Error Propagation in Programs[C]// IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE, 2018, pp. 279-290.
15. Sam Ainsworth, Timothy M. Jones. Parallel Error Detection Using Heterogeneous Cores[C]// IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE, 2018, pp. 338-349.
16. S.-B. Park and S. Mitra. IFRA: Instruction footprint recording and analysis for post-silicon bug localization in processors[C]// DAC, 2008, pp. 373-378.
17. A. DeOrio, Q. Li, M. Burgess, and V. Bertacco. Machine learning-based anomaly detection for post-silicon bug diagnosis[C]// DATE, 2013, pp. 491-496.
18. J. Carretero, X. Vera, J. Abella, T. Ramirez, M. Monchiero, and A. Gonzalez. Hardware/software-based diagnosis of load-store queues using expandable activity logs[C]// HPCA, 2011, pp. 321-331.
19. Dadashi M , Rashid L , Pattabiraman K , et al. Hardware-Software Integrated Diagnosis for Intermittent Hardware Faults[C]// IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE, 2014.
20. Jiaqi Yan, Guanhua Yan, Dong Jin. Classifying Malware Represented as Control Flow Graphs using Deep Graph Convolutional Neural Network[C]// IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE, 2019, pp. 52-63.
21. Deng Guanqian, Jing Qiu, Liu Guanjun, et al. A Discrete Event Systems Approach to Discriminating Intermittent from Permanent Faults[J]. Chinese Journal of Aeronautics, 2014, 27(2): 390-396.
22. Deng Guanqian, Jing Qiu, Liu Guanjun, et al. A Stochastic Automaton Approach to Discriminate Intermittent from Permanent Faults[J]. Journal of Aerospace Engineering, 2014, 228(6): 880-888.
23. Maurice G, Diaz F, Coti C, et al. Downtime statistics of current cloud solutions, 2014, http://iwgcr.org/wp-content/uploads/2014/03/downtime-statistics-current-1.3.pdf
24. Baraza J C, Gracia J, Gil D, et al. A Prototype of a VHDL based Fault Injection Tool: Description and Application [J]. Journal of Systems Architecture, 2002, 47 (10) : 847-867.
25. Zarandi H R, Miremadi G, Ejlali A R. Fault Injection into Verilog Models for Dependability Evaluation of Digital Systems[C]// Proceedings of the International Symposium on Parallel and Distributed Computing, IEEE Press, 2003: 281-287.
26. Sieh V, Tschche O, Balbach F. VERIFY: Evaluation of Reliability Using VHDL Models with Embedded Fault Descriptions[C]// Proceedings of the 27th International Symposium on Fault-tolerant Computing. Seattle, USA: IEEE Press, 1997: 32-36.
27. Chao Wang, Zhongchuan Fu, Hong-Song Chen, Gang Cui. Characterizing the Effects of Intermittent Faults on a Processor for Dependability Enhancement Strategy[J]. The Scientific World Journal, 2014:1-12.
28. Chao Wang, Wei Zhang. Intermittent fault injection platform implemented in register transfer level[J]. Journal of Beijing Information Science & Technology University, 2015 (30): 46-50.
29. Rashid L., Pattabiraman K., Gopalakrishnan S.. Characterizing the Impact of Intermittent Hardware Faults on Programs[J]. IEEE Transactions on Reliability, 2015, 64(1):297-310.
30. Radha Venkatagiri, Khalique Ahmed, Abdulrahman Mahmoud, Sasa Misailovic, Darko Marinov, Christopher W. Fletcher, Sarita V. Adve. gem5-Approxilyzer: An Open-Source Tool for Application-Level Soft Error Analysis[C]// IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE, 2019, pp. 214-221.

31. Qian Hu, etc al. Simics-based System Level Fault Injection Platform[J]. Computer Engineering, 2015(41):57-63.
32. Le M and Tamir Y. Fault injection in virtualized systems-challenges and applications. IEEE T Depend Secure 2015, 12(3): 284-297.
33. Chao Wang, Zhongchuan Fu. Quantitative evaluation of fault propagation in a commercial cloud system[J]. International Journal of Distributed Sensor Networks, 2020,16(3):1-11.
34. Hochreiter S , Schmidhuber J . Long Short-Term Memory[J]. Neural computation, 1997, 9(8):1735-1780.
35. M. L. Li, P. Ramachandran, S. Sahoo, S. Adve, V. Adve, and Y. Zhou. Trace based microarchitecture-level diagnosis of permanent hardware faults[C]//IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE, 2008, pp. 22-31.

Wang Chao is an Assistant Professor at the School of Computer Science in Beijing Information Science and Technology University of China. His main research interests are directed to reliability qualification of cloud computing using simulation and fault injection, and high performance computing in deep learning model accelerator development. He is also interested in unmanned ground vehicle and system, including objection detection, navigation and decision making.

Fu Zhongchuan is a vice professor in Computer Science department in Harbin Institute of Technology of China. His main research interests involve fault injection simulator development, and quantitative analysis of reliability technology in cloud computing environment. He is also interested in high performance computing in CPU model development.

Huo Yanyan is an algorithm engineer graduated from the School of Computer Science in Beijing Information Science and Technology University of China. She is interested in deep learning and AI area.

Received: June 20, 2020; Accepted: November 15, 2020.