Omission Approach of SDN-FIT evaluation to evaluate wide area distributed applications*

Hiroki Kashiwazaki[2,1,a)] Hiroki Takakura[2,b)] Shinji Shimojo[1,c)]

Abstract: A wide-area distributed application is affected by network failure due to natural disasters because the servers on which the application operates are distributed geographically in a wide area. Failure Injection Testing (FIT) is a method for verifying fault tolerance of widely distributed applications. In this paper, by limiting network failures only to the connection line, whole FIT scenarios are generated, and exhaustive evaluation of fault tolerance is performed. The authors propose a method to omit the evaluations from the aspect of topological constraint conditions. And they evaluate the visualization method of performance data obtained from this evaluation and the reduction of the fault tolerance evaluation cost by the proposed method.

1. Introduction

Information and communication services are essential to people's lives[*1]. There are various services using information communication technology (ICT), such as e-mail, map services. One of the reasons for the wide spreading of ICT services is the rapid spread of electronic devices such as smartphones. Cloud computing is one of the typical information communication services. Cloud computing is a form of provision of computers that can use computers without being aware of the location and number via the Internet.

Cloud computing is a concept advocated by Eric Schmidt, who was then CEO of Google in 2006 and has since advanced rapidly into research and development and commercial deployment. At present, Amazon Web Services (AWS) provided by Amazon, Microsoft Azure provided by Microsoft, Google Cloud Platform (GCP) provided by Google, and IBM Cloud provided by IBM, etc. are representative. They are known as a cloud computing service. Cloud computing services are still rapidly spreading, and the market is expanding. In fact, according to the domestic public cloud service market forecast announced by IDC Japan in October 2018, the domestic public cloud service market in 2018 is expected to increase 27.4% over the previous year to 666.3 billion yen.

*JSPS KAKENHI Grant Number 19K20256
1 Osaka University, Ibaraki, Osaka, 567–0047, Japan
2 National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
a) reo_kashiwazaki@nii.ac.jp
b) takakura@nii.ac.jp
c) shimojo@osaka-u.ac.jp
*1 Information and Communications in Japan WHITE PAPER 2018
`http://www.soumu.go.jp/johotsusintokei/whitepaper/eng/WP2018/2018-index.html`

From the viewpoint of ICT, considering the necessity of ICT services today, it is necessary to be able to always provide services with the same level of performance as in normal conditions. However, it is difficult to maintain the same level of performance under the condition of failure. The ICT service providers have become to be required to show users the service level agreement of their services. As to network service providers, it is also important to show the fault tolerance performance of the provider network. In other words, it is necessary for the ICT service provider to find how their ICT service can provide the performance under not only the normal conditions but also the non-steady conditions due to various failures.

2. Related Works

Failure Injection Testing (FIT) is widely known as a method to evaluate the fault tolerance of a service. FIT is an evaluation method that measures the quality of service when a failure occurs by intentionally injecting failures into the system that constitutes the service. Depending on the implementation environment, FIT can be roughly classified into two approaches, one is implemented in a production environment, and the other is implemented in a test environment. A representative example of FIT is Chaos Engineering proposed and developed by Netflix. In Chaos Engineering, administrators define the steady-state behavior of the system using externally observable performance indexes. Then, the behavior under non-steady condition is implemented by injecting the failures that assumed the stop of the server, the abnormal state of the hard disks, the disconnection of the network cable, etc. The fault tolerance of the system is evaluated by comparing the steady-state behavior to the non-steady one. One of the advantages of Chaos Engineering is that it is possible to implement traffic patterns and load patterns of actual services by performing fault tolerance evaluation in a production environment. Meanwhile, since the failure is injected into the production environment, it is necessary to minimize the influence of the failure injection not to degrade the level of the service.

Netflix has developed many automation tools and released it as open-source software to realize the above Chaos Engineering. Chaos Monkey is a tool to implement server failures by stopping virtual machines running on AWS at random. Besides, Chaos Gorilla is a tool to shut down all virtual machines running on a specific availability zone in AWS. Furthermore, Chaos Kong is a tool to stop all virtual machines in a particular region.

By implementing FIT using these tools usually, Netflix has built a wide-area distributed system with excellent fault tolerance and guarantees high service level agreement. Meanwhile, wide-area distributed systems have various network topologies depending on the arrangement of computer resources that compose them. In addition, when the number of locations that compose the wide-area distributed system increases, the combination of failures occurring on each network connection lines increases exponentially. When the fault tolerance evaluation is performed manually, it takes a lot of time and effort. So fault tolerance evaluation should be executed automatically.

DESTCloud is a platform for verifying and evaluating disaster tolerance and fault tolerance of wide-area distributed systems[*2]. DESTCloud uses Software Defined Disaster Emulation (SDDE) to inject network failure and collect logs generated during faults based on a disaster scenario described by the administrators who want to perform verification and evaluation of the system. In advance, the administrators describe the disaster scenario where kinds of failures are indicated in chronological order. Then the SDDE automatically injects the failure based on the disaster scenario into the network device using the Software-Defined Networking (SDN) approach. In addition, SDDE assigns a disaster scenario-specific ID to the log generated during failure occurrence. As a result, it is easy for administrators to analyze the log without any manual operations. However, a disaster scenario can be only described as a simple combination of failures, and it is not suitable for applications that try whole combinations of network failures. Also, it can not reduce the time and effort of analyzing logs for each combination.

The service quality of the ICT service cannot be found only by performing the benchmark once. It is necessary to acquire data comprehensively by changing multiple parameters that become indexes. Because of this kind of data acquisition, fault tolerance cannot be evaluated just by listing the data. Therefore, it makes sense to visualize the consolidated data. This study aims to propose a tool that automatically performs fault tolerance evaluation from data acquisition to visualization.

*2 DESTCloud Project
`https://ricc.itrc.net/DESTCloud`

The authors have already proposed an SDN-FIT system, a mechanism that generates programmable faults in the network that connects the FITs between bases and Availability Zones, not in VM units [1]. This proposal enumerates all combinations (failure scenarios) of failures that occur in network topologies that make up wide-area distributed applications and implements a comprehensive evaluation by implementing all of them. Using CLOUDIAN HYPERSTORE[*3], a globally distributed object storage, and COSBench, its benchmark software, all normalized I/O performance is listed. Thereby, the service manager can discover the weak point of the service at the time of failure. However, this method has a problem that the total number of failure scenarios increases exponentially as the number of nodes and links increases. In this study, we explain the SDN-FIT system and its design and propose and evaluate the failure scenario reduction method based on the requirements of the application.

3. Proposed Approach

In this paper, an ICT service provided by a geographically separated group of computers connected via a network is defined as a wide-area distributed service. The sites are connected by a route control device (router), and by operating this route control device, it is possible to generate intentional failures between the sites. The routers mean not only appliance products with physical enclosures, but also software routers installed on computers using x86 processors, and virtual routers that can be installed as virtual machines (VMs).

Connect to the console of the Network Operation System (NOS) that operates the router, and execute the NOS command using the Command Line Interface (CLI) to connect the routers among the sites. However, NOS commands may require interactive input. This interactive input requirement can be a barrier when trying to implement programmatic automation.

With the spread of cloud computing[*4], NOS also implemented cloud-like function sets when the cloud computing environments become possible to manipulate VM deployment and configuration changes using an application programming interface (API). In 2008, Cisco Systems in the United States released the API of its integrated router, Cisco ISR series, in 2008[*5]. For example, Vyatta, implemented as a software router, has implemented API operations since Ver. 6.2 in 2011. In this study, the authors evaluate the fault tolerance and automate this evaluation by generating intentional failures in the network connecting the sites using the API provided by NOS.

*3 `https://cloudian.com/jp/products/`
*4 The Internet White Paper [Japanese]
`http://www.impressrd.jp/news/180209/NP`
*5 `https://tech.nikkeibp.co.jp/it/article/NEWS/20080528/304536/`

3.1 Classification of network failures

In a FIT, it is assumed that network failure caused by a natural disaster can be implemented. There are various factors in the network failure caused by a natural disaster, and those are, failures directly caused by natural disaster and the in-direct failure caused by equipment failure, etc. In addition, it is also necessary to examine the influence range of the failure pattern, the presence or absence of spatial change, and the temporal transition. The network failures that are caused by a natural disaster can be classified according to reports of the Ministry of Internal Affairs and Communications "Study Group on the Ways to Secure Communications in Large-scale Disasters and Other Emergency Situations" and "Information Network Safety and Reliability Standards"[*6] and "Information Network Safety and Reliability Standards"[*7]. The reports show faults for communication equipment etc. and classify control applied to network equipment for each event (Table 1).

cause of disorder	disorder factor	presentation	function to be implemented
control operation or software	communication restriction control	congestion	latency and n% packet loss
			traffic shaving
	illegal route advertisement	loop of routes	RIB/FIB force alter
		flapping of routes	
		route disorder (unknown destination)	
network equipment	disorder of equipment (entire)	communication lost (entire)	interface down
	disorder of equipment (part)	communication lost (part)	
	over load of equipment	packet loss	n% packet loss
		rise latency time	add latency
communication line	cable discoonection	communication lost (part)	interface down and 100% packet loss
	disorder of repeater/switch		
	concentrate of traffic	congestion	latency and n% packet loss
			traffic shaving
facility	destroy of office	communication lost (entire)	interface down and 100% packet loss
	lost of power supply		
	disorder of cooler		
		communication lost (part)	

Table 1 Classification of network failures

From the aspect of "cause of disorder," causation can be assumed in control operation or software, network equipment, communication line. From the aspect of "disorder factor", causation on control operation or software can be caused by disorders of communication restriction control and illegal route advertisement. Disorders on network equipment can be caused by entire/partial equipment and overload of equipment. Disorders on communication lines can be caused by cable disconnection, a disorder of repeaters or switches and concentrate on traffic. Finally, disorders on the facility can be caused by the destruction of office, lost of power supply, and disorder of cooler. These factors can be presented by the phenomenon of congestion, loop or flapping of routes, communication lost, packet loss, and rise of latency time. This classification can result in the network function required to be implemented in enough evaluation of fault tolerance. The requirement is shown below.

(1) Increased delay
(2) n% packet loss ($0 < n \leq 100$)
(3) Deactivate network interface card (NIC)
(4) Change of routing control table

[*6] http://www.soumu.go.jp/main_sosiki/kenkyu/saigai
[*7] http://www.soumu.go.jp/menu_seisaku/ictseisaku/net_anzen/anshin

Therefore, we set the four types of obstacles implemented in this research.

3.2 Proposed system

Figure 1 shows a schematic diagram of the proposed system for implementing fault tolerance verification by intentional failure occurrence and its automation. The system consists of a failure pattern generator, FIT controller, benchmark controller, and visualizer. The failure pattern generator generates whole failure patterns based on the topology of the wide-area distributed system. The FIT controller inputs the failure pattern and implements the failures to SDN routers. The controller also always collect SDN router information and maintain topology information of the wide-area distributed system. The benchmark controller performs the benchmark program on the wide-area distributed system cooperated with the FIT controller. After that, the benchmark controller sends the benchmark result to the visualizer. The visualizer receives and put the measurement results in order, then visualizes the data. The following sections describe each component.

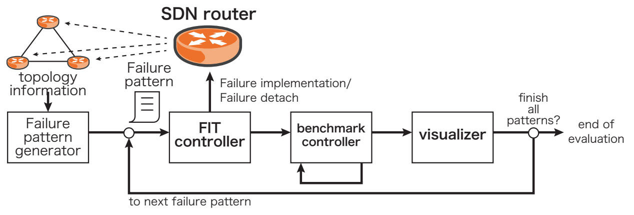

Fig. 1 A diagram of proposed SDN-FIT system

3.2.1 Failure pattern generator

The fault pattern generator generates fault patterns according to the number of circuits in the topology from the topology information of the wide-area distributed service. The sites supporting wide-area distributed services to be subjected to fault tolerance verification are connected by a routing controller that can be operated by API. The identifiers are given to each base, and the NICs at both ends of the circuit connecting the sites are given identifiers in the NOS of each router. From the above information, the topology of the site supporting wide-area distributed service can be expressed by the nesting of hash and array. Yet Another Markup Language (YAML) is a format that represents structured data, and the topology can be described using YAML. For example, a network consisting of three sites in Figure 1 can be expressed, as shown in Listing 1. In this topology data, site A is connected to B by eth0 and to C by eth1; site B is connected to A by eth0 and A by eth1; and site C is connected to A by eth0 and B by eth1. Listing 1 is an example of a YAML file to indicate it.

Listing 1

3nodes_network

```
- A:
- [[eth0, B], [eth1, C]]
- B:
  - [[eth0, A], [eth1, C]]
  - C:
  - [[eth0, A], [eth1, B]]
```

At the same time, this topology data shows the circuit between sites. In the example of Listing 1., the line a connecting eth0 of site A to eth0 of site B, the line b connecting eth1 of site A to eth0 of site C, and the line connecting eth1 of site B to eth1 of site C Indicates that there are three lines of c. When the number of lines is m, the fault pattern generator searches for combinations of fault patterns that generate all $n(0 < n \leq m)$ double faults in each line. One failure pattern is represented by an array composed of the failure type identifier, the identifier of the router that generates the failure, and the identifiers of one or more NICs that cause the failure in the router. Listing 2. shows the case where the line a and the line b are interrupted due to the deactivation of the NIC.

Listing 2
shut-
down

```
- [shutdown, A, eth0, eth1]
```

3.2.2 FIT controller

The FIT controller uses the fault patterns created by the fault pattern generator to update probabilistic data in accordance with each fault. The implementer of fault tolerance verification provides the FIT controller with router information of the site supporting the wide-area distributed service to be verified. The FIT controller uses the API for the router to obtain NIC information of each router and the IP address assigned to that NIC. It is determined that NICs in the same IP address range at different sites are connected, and topology data is created.

The FIT controller provides the created topology data to the fault pattern generator, and the fault pattern generator returns all fault patterns to the FIT controller. The FIT controller sequentially processes the obtained fault pattern data. As described in Section 3.2.1, a failure pattern consists of an identifier of the failure type, an id of the router that causes the failure, and an identifier of one or more NICs that cause the failure in that router. The FIT controller reads this array and uses the API to control the NIC specified as the router and the instruction corresponding to the identifier of the failure type.

After the control that implements the fault condition usually ends, the FIT controller applies to process to the benchmark to measure the performance in the event of a fault. When the execution of the benchmark usually ends, the FIT controller controls the specified NIC of the router using the API and cancels the failure status. When the release of the fault condition usually ends, the FIT controller applies to process to the visualizer to visualize the performance measurement results obtained by the benchmark controller. Execute these processes for all failure patterns, and repeat them until finished.

3.2.3 Benchmarker

Benchmark controller performs object storage benchmarking. The benchmark controller then sends the benchmark results to the visualizer. In benchmark controller, benchmark software is implemented according to the wide-area distributed service to be verified. According to an instruction from the FIT controller, benchmark controller executes the specified benchmark software based on the specified arguments.

Those who perform fault-tolerance verification install benchmark software according to the items they want to investigate. For example, if the wide-area distributed service is a Web service and you want to verify its response performance, the fault-tolerant verifier uses Apache Bench[*8]. If wide-area distributed services are POSIX compliant storage, fio[*9] or IOZONE[*10] may be used as benchmark software.

3.2.4 Visualizer

The visualizer receives measurement results from the benchmark controller and visualizes the result data. The measurement results obtained by the benchmark controller are placed in a local storage area in the computer where the visualizer is deployed, or placed in a place that can be obtained by remote access. When the visualizer receives an instruction from the FIT controller, it reads the specified file and visualizes the data according to the specified drawing method. The visualizer shows the location of the visualized file. This enables the verifier to view the visualized data.

4. Implementation

Authors deployed a wide-area distributed service in a real environment and implemented SDN-FIT system to verify the fault tolerance of this wide-area distributed service.

4.1 Implementation of evaluation environment

4.1.1 Distcloud

Distcloud is a wide-area distributed virtualization platform under Regional InterCloud Subcommittee (RICC) of the Internet Technology 163rd Committee (ITRC) of the Japan Society for the Promotion of Science and Technology. It is constructed by connecting computers distributed by geographically dispersed universities, research organizations, and cloud computing providers by broadband networks (Figure 2). Wide-area distributed virtualization infrastructure is implemented by deploying scale-out distributed storage. Focusing on live migration as a disaster recovery method, authors implement storage technology with little degradation of I/O performance before and after wide-area live migration [2], [3].

Distcloud's sites are connected by SINET[*11], an academic information network provided by the National Institute of Informatics. It uses L2VPN/VPLS service[*12] that allows

[*8] `https://httpd.apache.org/docs/2.4/programs/ab.html`
[*9] `https://github.com/axboe/fio`
[*10] `http://www.iozone.org/`
[*11] `https://www.sinet.ad.jp/`
[*12] `https://www.sinet.ad.jp/connect_service/service/l2vpn`

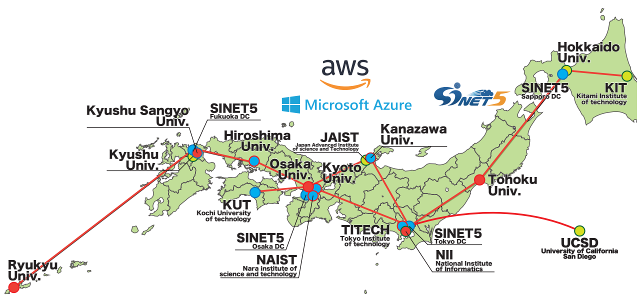
aws
Microsoft Azure
SINET5
Hokkaido Univ.
KIT Kitami Institute of technology
SINET5 Sapporo DC
Kyushu Sangyo Univ.
SINET5 Fukuoka DC
Hiroshima Univ.
JAIST Japan Advanced Institute of science and Technology
Kanazawa Univ.
Kyushu Univ.
Osaka Univ.
Kyoto Univ.
Tohoku Univ.
KUT Kochi University of technology
TITECH Tokyo Institute of technology
SINET5 Tokyo DC
SINET5 Osaka DC
UCSD University of California San Diego
NAIST Nara Institute of science and technology
NII National Institute of Informatics
Ryukyu Univ.

Fig. 2 Schematic Diagram of Distcloud (2019)

Ethernet frames to be exchanged between LANs at remote sites.

Virtual Private LAN Service (VPLS) [4] is a technology that can transfer Ethernet frames using Multi-Protocol Label Switching (MPLS) defined in RFC3031 [5]. Because a virtual Ethernet LAN can be constructed for each network created in each network, a protocol to be used does not depend on IP, and a network with L2 connectivity can be constructed (Figure 3).

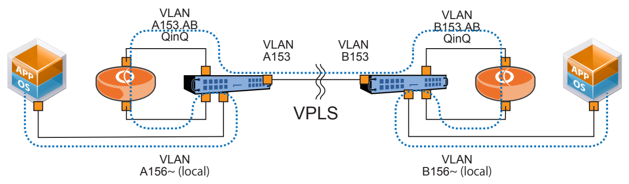

Fig. 3 Inter-communication among sites with VPLS

Distcloud uses SINET VPLS and prepares L2 networks called distcloud-core and distcloud-mgmt, respectively. A distcloud-core is a network used for communication of services and applications, and a distcloud-mgmt is a network for management of devices constituting the sites. Besides, an L3VPN network called distcloud-L3 is prepared separately. As for distcloud-L3, /24 IPv4 addresses are assigned in advance for each site.

A site connected to Distcloud needs to prepare a VLAN to connect with the distcloud-core, distcloud-mgmt and distcloud-L3 in the LAN of the site. A site connected to Distcloud prepares computer resources and connects this with the VPLS as mentioned above. Two L2VPN/VPLS connectivity by VPLS provided by SINET, one IPv4 network by L3VPN, three VLANs in the site, and computer resources connected to it is the environment provided by Distcloud.

4.1.2 VyOS

VyOS[*13] is a network OS developed by open source. It is developed based on Debian GNU / Linux. Formerly from Vyatta mentioned in section 3, it was forked from version 6.6 R1 of Vyatta Core, which is the free version of Vyatta. In addition to being installed on a physical computer and used as a software router, it may also be installed as a VM in a virtual environment and used as a virtual router. Like a general NOS, it has a unified CLI like a hardware router.

[*13] https://vyos.io/

In order to cause communication failure due to FIT proposed in this research among sites, it is necessary to configure an independent network at each site that configures Distcloud, and it is necessary to perform routing control with the deployed router at the site. The NICs connecting between the sites are independent of the networks owned by each site, and the two connected sites need to belong to the same network. In Distcloud, only the aforementioned network with distcloud-core is provided as a service network.

4.1.3 CLOUDIAN HYPERSTORE

CLOUDIAN HYPERSTORE is an object storage product that is fully compatible with the Amazon S3 API[*14] marketed by CLOUDIAN.

Object storage is a computer data storage that manages data as an object as opposed to filesystems that manage data as a file hierarchy and other storage architectures such as block storage that manages data as blocks specified by sectors and tracks Refers to the architecture. Each object contains data, metadata, and a unique identifier. Object storage can be implemented at multiple levels, including object storage device level, system level, and interface level, in which case object storage is an interface directly programmable by the application, multiple instances of physical hardware It provides data management functions including namespaces that can span and replication of data.

4.1.4 COSBench

COSBench is an object storage benchmark tool developed by Qing Zheng et al. Object storage has different indexes (workloads) to keep the performance of the access system in a proper state for each service that utilizes it. However, in 2013, when the use of object storage started to increase worldwide, there was no workload for object storage. COSBench was designed and implemented to address this problem [6].

4.2 Construction of wide area distributed system environment

In this study, CLOUDIAN HYPERSTORE and its environment for verification are constructed using five Distcloud sites (Osaka University, Tohoku University, Ryukyu University, National Institute of Informatics, and Kyushu Sangyo University) (Figure 4). The specifications of machines installed in each site is shown in Table.2. The ones of networks installed in each site is shown in Table.3.

Install Ubuntu[*15] 18.04 LTS, an operating system based on Debian GNU/Linux, on the x86 server at each site. To run VM on this Linux, authors build the environment of KVM[*16] which is a virtualization module that makes Linux kernel function as a virtual hypervisor.

Then the authors created the following four VMs on Linux installed on the x86 server at each site.

- CLOUDIAN HYPERSTORE 2VMs
- CentOS7 for COSBench 1VM

[*14] https://aws.amazon.com/jp/s3/
[*15] https://www.ubuntu.com/
[*16] http://www.linux-kvm.org/page/Main_Page

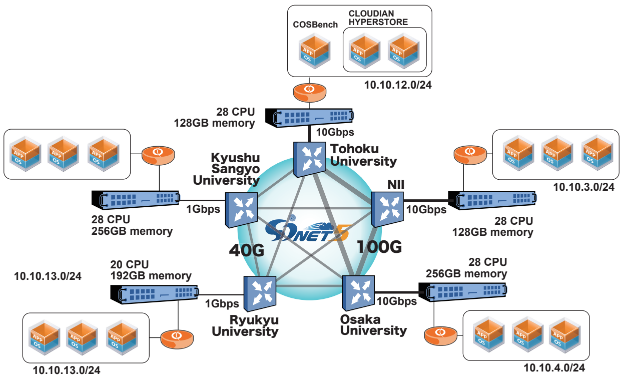

Fig. 4 A diagram of the wide-area distributed system

site name	number of physical cores	capacity of main memory (MB)	capacity of storages (GB)
Osaka University	28	256	3,600
Tohoku University	28	128	2,200
Ryukyu University	20	192	2,200
Kyushu Sangyo University	28	256	3,300
NII	28	128	1,400

Table 2 The specifications of machines installed in each site.

site name	network bandwidth to SINET (Gbps)	network bandwidth in the site (Gbps)
Osaka University	100	10 (dedicated)
Tohoku University	100	10 (best effort)
Ryukyu University	40	1
Kyushu Sangyo University	10	1
NII	100	10 (dedicated)

Table 3 The specifications of networks installed in each site.

- VyOS 1VM

The VMs specifications of CLOUDIAN HYPERSTORE, COSBench, and VyOS are shown in Table 4.

	Cloudian Hyperstore	COSBench	VyOS
OS/ Version	CentOS Linux release 7.4.1708	CentOS Linux release 7.6.1810	VyOS 1.1.8
RAM [MiB]	32,768	4,096	512
number of vCPUs	8	2	1

Table 4 VM Specifications

The VMs belong to an independent network for each location and assigns an IPv4 address that does not overlap with the networks of other locations. A unique VLAN is assigned to this network in the site, and VMs for CLOUDIAN HYPERSTORE at each site, VMs for COSBench, and one NIC of VyOS are connected to the bridge interface of this VLAN.

The VyOS at each site has a NIC for configuring a backbone network connected to the VyOS at the other sites. As described in Section 4.1.2, NICs connected to each backbone network need to belong to independent VLANs, so VLANs ID should be selected not to overlap with other VLANs at all sites. The two NICs connected to the backbone network are connected via a unique L2 network created on the L2 network of distcloud-core by Q-in-Q.

In VyOS at each site, OSPF [7] is operated as an Interior Gateway Protocol [8], the cost with the adjacent site is set to 10, dead-interval to 40 seconds, hello-interval to 10 seconds, and retransmit-interval to 5 seconds. In this way, VMs belonging to the networks at each site can communicate with each other. Also, by setting disabled for the interconnected NICs, that NIC can be deactivated, and communication disconnection can occur. When the NIC becomes inactive, and communication interruption occurs, OSPF recalculates the path in the topology where communication interruption occurred, and the path is changed by sending Link State Update packet. The inactive state of the NIC can be released by the delete command.

4.3 Omission of evaluation

As the number of nodes increases, the number of links also increases depending on the topology. For example, in a full mesh topology network with n nodes, the number of links is $\frac{n(n-1)}{2}$, and increases in the order of a square with respect to the increase in n. When the number of links is l, the maximum number of link failures in the topology is assumed to be $l(l > 0)$. Also, the total number of link failures for all l major failures is $2^l - 1$ (2^l if $l = 0$ is included, but $l = 0$ is a steady state in which no link failure has occurred). Thus, the total number of failure scenarios increases geometrically with the increase in the number of nodes, particularly the number of links. Therefore, when all failure scenarios are implemented and evaluated, the total time required for the evaluation also increases geometrically. Therefore, it is necessary to reduce the failure scenarios to be evaluated with a reasonable method.

When a link failure occurs in the topology in the steady-state, the number of network edges changes. A group of nodes that can reach each other among the nodes belonging to the network is called as a "cluster". A network that was one cluster in a steady-state may become a plurality of separated c clusters due to occurrence of n link failures. As the number of clusters increases, the state of the network changes significantly, and the behavior of applications that run on it also differs. Therefore, it is meaningful to set a constraint condition of the topology required by the application and evaluate only a failure scenario that matches the condition even if a link failure occurs. This is because a failure scenario that is not so is an operation outside the definition of the application. However, it is important to indicate what percentage of the total number of failure scenarios is a failure scenario that results in undefined operation.

In this way, the failure scenario to be implemented and the failure scenario to be omitted without being implemented are classified by comparing the number of clusters in the network topology according to the failure scenario and the constraints required by the application. This approach can provide effective suppression against a geometrical increase in the total number of failure scenarios.

5. Evaluations

Authors evaluate the disaster tolerance of CLOUDIAN HYPERSTORE quantitatively using CLOUDIAN HYPERSTORE, which is constructed in Section 4.2 and the SDN-FIT system implemented. Then the authors visualize the results of evaluations. The network consisting of five sites constructed in Section 4.2 is connected by ten independent connection lines, and the failure pattern of a single failure on the network is also ten patterns.

multiplicity of failures	number of failure patterns	number of clusters				
		1	2	3	4	5
1	10	10	0	0	0	0
2	45	45	0	0	0	0
3	120	120	0	0	0	0
4	210	205	5	0	0	0
5	252	222	30	0	0	0
6	210	125	85	0	0	0
7	120	0	110	10	0	0
8	45	0	0	45	0	0
9	10	0	0	0	10	0
10	1	0	0	0	0	1

Table 5 Classification of failure patterns with number of clusters

Since this evaluation experiment network consists of ten connection lines, a maximum of ten faults will occur. Since all nodes are connected to other nodes by four lines, the situation where the number of clusters is greater than 1 arises from a quadruple failure. Table 5 shows the number of simultaneous failures, the number of failure patterns, and the number of failure states belonging to each cluster number (maximum number = 5).

CLOUDIAN HYPERSTORE can set a duplication policy for each bucket. In response to one file creation request, if you set a policy to return ACK by creating two duplications on all nodes, the number of clusters=1 is necessary. Meanwhile, a failure scenario with two or more clusters will be undefined, and CLOUDIAN HYPERSTORE will return an error in response to the request. Therefore, the benchmark cannot be executed in these scenarios. That is, It can be said that CLOUDIAN HYPERSTORE can perform the in-definition operation with the total number of failure scenarios 801 with 1 cluster for all 1,023 failure scenarios, but the remaining 422 is undefined and cannot be evaluated. This is not only a number, but it is necessary to evaluate the probability of multiple failures.

6. Conclusion

In this paper, authors proposed a system that supports automation of fault tolerance evaluation of wide-area distributed service for the purpose of quantifying fault tolerance evaluation of information communication service constructed as a wide-area distributed system and reducing the cost required for evaluation. This system consists of a fault pattern generator, FIT controller, benchmark controller, and visualizer.

As the number of nodes and links increases, the total number of failure scenarios increases exponentially. Therefore, by comparing the operation requirements of the application with the topology after the failure scenario is implemented, we propose to reduce the number of failure scenarios that are actually implemented and evaluate when evaluating the CLOUDIAN HYPERSTORE on a 5-node full mesh network. It was shown that the total number could be reduced by 22%. Further reductions can be expected in applications with more complex requirements.

On the other hand, it is required not only to evaluate the reduction in the number of failure scenarios to be evaluated simply but also to evaluate the resilience after evaluating the occurrence probability of multiple failures and the number of reductions. This is an issue for the future.

In order to evaluate the effectiveness of this proposal, the authors constructed a wide-area distributed service on the wide-area distributed platform “Distcloud” and performed fault tolerance verification by implementing the proposed method for this service. Perform comprehensive benchmarks based on failure patterns that are automatically generated by providing router information at multiple locations, and compare the steady-state and non-steady-state performances to reduce the performance against steady-state The heat map was output and visualized.

Acknowledgments Part of this work was carried out under the Cooperative Research Project Program of the Research Institute of Electrical Communication, Tohoku University. The research was supported by ROIS NII Open Collaborative Research 19FA08. The work is partly supported by the collaborative research program 2019 information initiative center, Hokkaido university, Sapporo, Japan. Thanks for Cloudian Inc. They always provided technical support of CLOUDIAN HYPERSTORE on the Distcloud environment.

References

[1] Kashiwazaki, H., Miura, S. and Shimojo, S.: A Proposal of SDN-FIT System to Evaluate Wide-Area Distributed Applications Based on Exhaustive FIT Scenario Generation, *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2, pp. 36–41 (online), DOI: 10.1109/COMPSAC.2019.10180 (2019).

[2] Nakagawa, I., Ichikawa, K., Kondo, T., Kitaguchi, Y., Kashiwazaki, H. and Shimojo, S.: Transpacific Live Migration with Wide Area Distributed Storage, *2014 IEEE 38th Annual Computer Software and Applications Conference*, pp. 486–492 (online), DOI: 10.1109/COMPSAC.2014.71 (2014).

[3] Nakagawa, I., Kashiwazaki, H., Shimojo, S., Ichikawa, K., Kondo, T., Kitaguchi, Y., Kikuchi, Y., Yokoyama, S. and Abe, S.: A Design and Implementation of Global Distributed POSIX File System on the Top of Multiple Independent Cloud Services, *2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pp. 867–872 (online), DOI:

10.1109/IIAI-AAI.2016.75 (2016).

[4] Rekhter, Y. and Kompella, K.: Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling, RFC 4761 (2007).

[5] Viswanathan, A., Rosen, E. C. and Callon, R.: Multiprotocol Label Switching Architecture, RFC 3031 (2001).

[6] Zheng, Q., Chen, H., Wang, Y., Zhang, J. and Duan, J.: COS-Bench: Cloud Object Storage Benchmark, *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, New York, NY, USA, ACM, pp. 199–210 (online), DOI: 10.1145/2479871.2479900 (2013).

[7] Ferguson, D., Lindem, A. and Moy, J.: OSPF for IPv6, RFC 5340 (2008).

[8] Faucheur, F. L., Merckx, P., Telkamp, T., Uppili, R. and Vedrenne, A.: Use of Interior Gateway Protocol (IGP) Metric as a second MPLS Traffic Engineering (TE) Metric, RFC 3785 (2004).