Proceedings of the APAN – Research Workshop 2019
ISBN 978-4-9905448-9-8

Evaluation of wide-area distributed services by SDN-FIT system

Hiroki Kashiwazaki, Shinnosuke Miura, Hiroki Takakura and Shinji Shimojo

***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 to the connection line, whole FIT scenarios are generated and exhaustive evaluation of fault tolerance is performed. Authors evaluate the visualization method of performance data obtained from this evaluation and the reduction of the fault tolerance evaluation cost by the proposed method.**

***Index Terms*—Distributed Systems, Failure Injection testing, Resilience**

I. Introduction

INFORMATION and communication services are essential to people's lives. 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. 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.

In addition, the market size in 2022 is estimated to be 1.46.5 trillion yen, which is 2.8 times that in 2017. Cloud computing services are classified into Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), etc. according to the service level. SaaS is a form of cloud computing that provides the software. PaaS provides language processing systems, libraries, middleware, etc. as a basis for operating software. IaaS provides computing infrastructure such as CPU, memory, disk, and network. Furthermore, in recent years, a wide variety of cloud computing services such as Machine Learning as a Service (MLaaS), which provides machine learning services, and Desktop as a Service (DaaS), which provides personal desktop environments, are becoming widespread. is there.

The ICT services are built on a wide area distributed system composed of computer resources of multiple geographically dispersed sites for the purpose of load distribution and improvement of fault tolerance. By distributing geographically, robustness can be secured against failure at a single site. However, the ICT service constructed as a wide area distributed system can be vulnerable to the simultaneous multiple failures of the network lines. Various factors can cause network failure. For example, packet loss can be caused by network device failure, disconnection of a network cable and human error caused by incorrect operation. Also, especially in Japan where natural disasters occur frequently, network failures due to disasters are also conceivable. In fact, in the case of large-scale disasters represented by the Great Hanshin-Awaji Earthquake (1995) and the Great East Japan Earthquake (2011), communication path interruption was a threat.

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

This work was supported in part by JSPS KAKENHI Grant Number 19K20256.

Hiroki Kashiwazaki, Shinnosuke Miura and Shinji Shimojo are with the Osaka University, Japan (e-mail: reo, shinnosuke.miura, shimojo @ cmc.osaka-u.ac.jp, respectively).

Hiroki Kashiwazaki and Hiroki Takakura are with National Institute of Informatics, Japan (e-mail: reo_kashiwazaki, takakura@nii.ac.jp, respectively)

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 own 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 own ICT service can provide the performance not only under the normal conditions but also under the non-steady conditions due to various failures.

II. Related Works

Failure Injection Testing (FIT) is widely known as a method to evaluate the fault tolerance of a service. This method 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 the former is Chaos Engineering proposed by Netflix. In Chaos Engineering, first authors 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. On the other hand, 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 a number of automation tools and released it as open source software in order to realize the above Chaos Engineering. Chaos Monkey is a tool to implement server failures by stopping virtual machines running on AWS at random. In addition, Chaos Gorilla is a tool to stop all virtual machines running on a specific availability zone in AWS. Furthermore, Chaos Kong is a tool to stop all virtual machines in a specific 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. 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.

III. 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. Routers include 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).

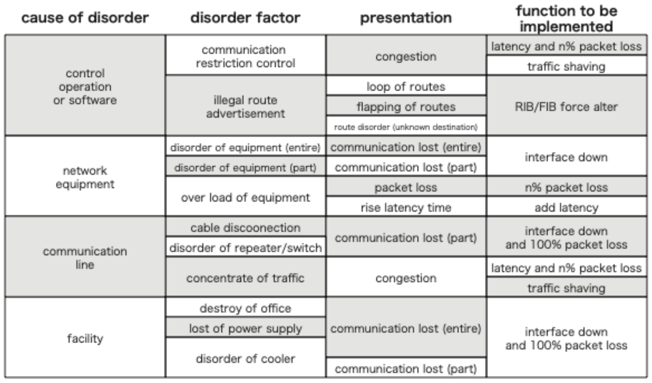
Connect to the console of the Network Operation System (NOS) that operates the router, and execute the NOS command using the Command Line Interface (CLI) to connect the routers among the sites. However, NOS commands may require interactive input. This interactive input requirement can be a barrier when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

With the spread of cloud computing, 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. 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.

A. 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, 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". The reports show faults for communication equipment etc. and classify control applied to network equipment for each event (Table 1).

From the aspect of "cause of disorder" causation can be



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

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

Therefore, the authors implement the four types of network failures in this research.

B. 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.

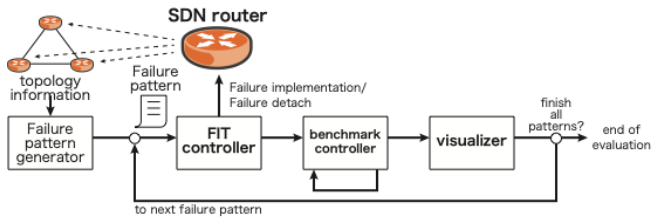


Figure 1 A diagram of proposed SDN-FIT system

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 YAML file to indicate it.

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

Listing 1.

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 3 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.

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

Listing 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 III-B1, a failure pattern consists of an identifier of the failure type, an identifier 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 ends normally, 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 ends normally, 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 ends normally, 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) *Benchmark controller*

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. If wide area distributed services are POSIX compliant storage, fio or IOZONE may be used as benchmark software.

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.

IV. Implementation

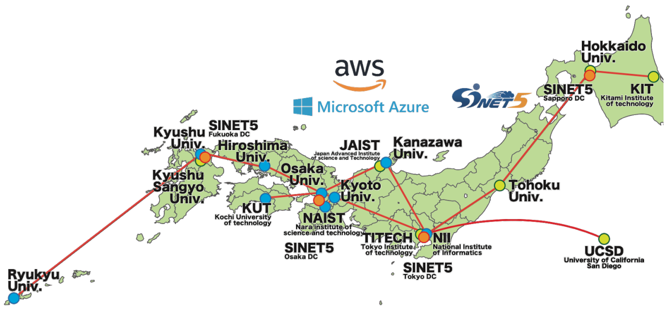
Authors deployed a wide area distributed service in a real environment and implement SDN-FIT system to verify the fault tolerance of this wide area distributed service.

A. *Implementation of the evaluation environment*

1) *Distcloud*

Distcloud is a wide-area distributed virtualization platform under Regional InterCloud Subcommittee (RICC) of the Internet Technology 16th 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.

Distcloud's sites are connected by SINET, an academic information network provided by the National Institute of



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

Figure 2. Schematic Diagram of Distcloud (2018)

Informatics. It uses L2VPN / VPLS service that allows Ethernet frames to be exchanged between LANs at remote sites.

Virtual Private LAN Service (VPLS) is a technology that can transfer Ethernet frames using Multi-Protocol Label Switching (MPLS) defined in RFC3031. 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).

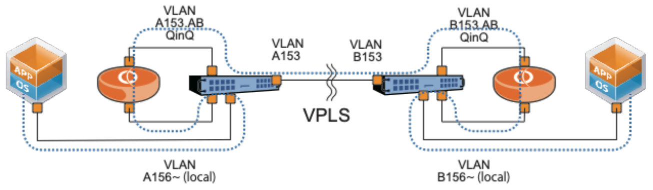


Figure 3. Inter-communication among sites with VPLS

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. In addition, 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 above-mentioned VPLS. 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.

2) VyOS

VyOS is a network OS developed by open source. It is developed based on Debian GNU / Linux. Originally from Vyatta mentioned in section III, 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.

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.

Although it is conceivable to newly secure an independent VLAN for connection between sites as L2VPN / VPLS, it is necessary to apply for the number of lines connecting between sites and to apply L2VPN / VPLS. This method becomes impractical if the number of connected lines increases. Therefore, by using IEEE802.1ad (Q-in-Q) in a router deployed at each site, networks of different VLANs can be configured across different sites on the distcloud-core network.

VyOS is a network OS that can communicate with Q-in-Q and can realize all the failure implementations described in Section III-A on its own. As VyOS is developed based on Debian GNU / Linux as mentioned above, it can be used by specifying the tc command of Linux as traffic-policy of VyOS. For these reasons, it is used for verification experiments of this study.

3) CLOUDIAN HYPERSTORE

CLOUDIAN HYPERSTORE is an object storage product that is fully compatible with the Amazon S3 API 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.

CLOUDIAN HYPERSTORE has a function to manage data at the bucket level, and control parameters can be defined at the bucket level. The bucket policy is a parameter that determines the number of copies of data. In this evaluation experiment, when the client uploads a file (PUT operation) when three copies are created at all sites. The policy is to return an acknowledgment (ACK). CLOUDIAN HYPERSTORE is a wide area distributed service that is also used in the back end of the video sharing site “Nico Nico Douga” of Dwango Co., Ltd.

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.

The development of COSBench, which has been developed by Intel, aims at preparing both object storage system performance comparison and system optimization and is a scalable implementation to cope with the scale of the system. At COSBench, there are two types of drivers: a driver that loads object storage, and a controller that instructs to load the driver. If the load details such as read / write (R: W) ratio are described in the XML file that describes the workload and registered in the controller by the web console or the command for CLI, it will be queued on the controller. The load test is performed sequentially.

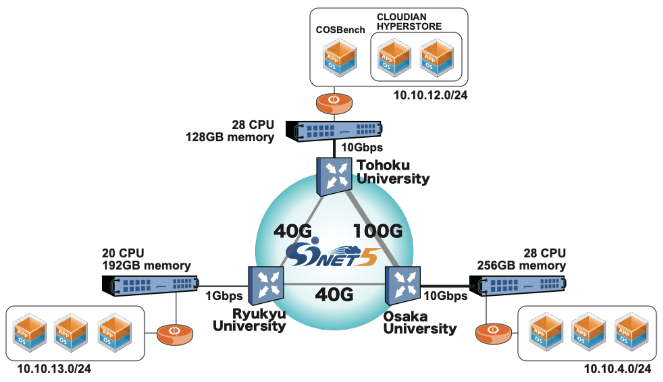


Figure 4. diagram of the wide-area distributed system

B. Construction of wide-area distributed system environment

In this study, CLOUDIAN HYPERSTORE and its environment for verification are constructed using three Distcloud sites (Osaka University, Tohoku University, Ryukyu University) (Figure 4). The x86 server installed at Osaka University has a CPU of 28 physical cores, a main memory of 256GB, a 3.6TB SSD array is connected, and an exclusive 10 Gbps leased line is connected to the campus network. Tohoku University has 28 physical cores of CPU, 128GB of main memory and 2.2TB of disk array connected and is connected to the campus network via a shared 10 Gbps line. SINET 5 connects Osaka University and Tohoku University at 100 Gbps, and Osaka University and Ryukyu University, and Tohoku University and Ryukyu University at 40 Gbps.

Install Ubuntu 18.04 LTS, an operating system based on Debian GNU/Linux, on the x86 server at each site. In order to run VM on this Linux, authors build the environment of KVM 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.

	OS/Version	RAM [MiB]	number of vCPUs
CLOUDIAN HYPERSTORE	CentOS Linux release 7.4.1708	32768	8
COSBench	CentOS Linux release 7.6.1810	4096	2
VyOS	VyOS 1.1.8	512	1

Table 2. Performance of VM for Cloudian Hyperstore, COSBench and VyOS

- CLOUDIAN HYPERSTORE 2VMs
- CentOS7 for COSBench 1VM
- VyOS 1VM

The VMs performance of CLOUDIAN HYPERSTORE, COSBench, and VyOS are shown in Table 2.

The VM belongs 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 two sites. As described in Section 3.1.2, NICs connected to each backbone network need to belong to independent VLANs, so select VLANs that do not overlap with 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 is operated as an Interior Gateway Protocol, 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.

C. Implementation of SDN-FIT system

In this research, five programs were created to implement the FIT controller, the benchmark controller, and the visualizer among the proposed systems described in Section III. In the FIT controller, in this paper, in order to simplify the evaluation of CLOUDIAN HYPERSTORE, authors implemented the deactivation of the network interface among the four faults shown in Section III-A. The outline of each script is as follows.

1) network failure implementation script

The network failure implementation script is one of the scripts that configure the FIT controller and causes a failure in the network connecting among sites. In VyOS, it is separated into operation mode and configuration mode, and it connects to the VyOS console and switches to configuration mode by entering configure at the prompt. Here, in order to deactivate eth0, you need to input command as follows.

```
# set interface ethernet eth0 disable
```

The network failure implementation script is an implementation of this series of processing using VyOS cli-shell-api. The network failure implementation script must first initialize the environment. Use a command to acquire environment variables required for initialization.

```
# /bin/cli-shell-api getSessionEnv
```

The getSessionEnv command outputs a series of operations specific to the session by giving a process identifier as an argument. The initialization is completed by executing as follows after initialization.

```
# /bin/cli-shell-api setupSession
```

After initialization is complete, the following command can deactivate the NIC with the identifier specified by the NIC identifier.

```
# /opt/vyatta/sbin/my_set ethernet [NIC identifier] disable
```

The program that executes this series of processing receives the identifier of NIC as an argument. When multiple NIC identifiers are specified, the specified NICs are sequentially deactivated.

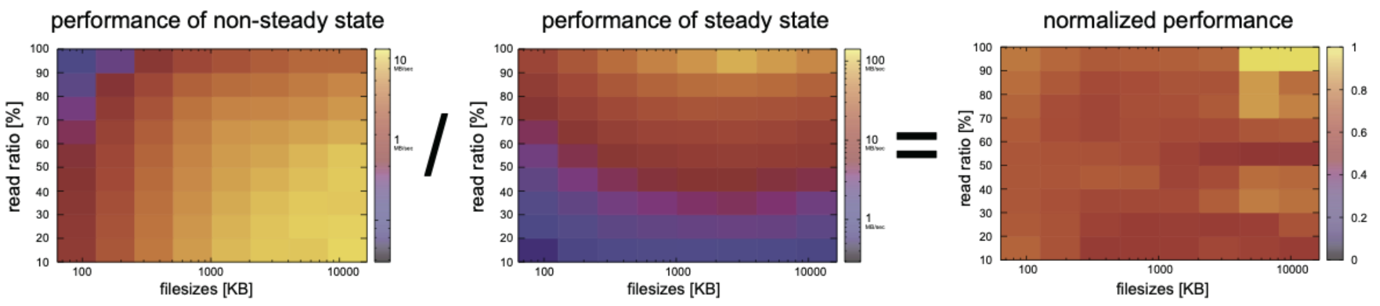


Figure 5. Calculation diagram of normalized performance.

2) network failure deactivation script
The network failure deactivation script is one of the programs that configure the FIT controller and is a script that deactivates the failure that has occurred in the network connecting the sites. As described in Section 3.3.1 Network Failure Implementation Script, after using cli-shell-api of VyOS and initializing by executing getSessionEnv command and /bin/cli-shell-api/setupSession, it is used as an argument. Execute the following command for the specified NIC identifier to release the inactive status.

```
# /opt/vyatta/sbin/my_delete ethernet [NIC identifier] disable
```

The network failure deactivation script can receive multiple arguments in the same way as the network failure implementation script, and when multiple NIC identifiers are specified, the inactive state of the specified NIC is released sequentially.

3) Benchmark execution script
The benchmark execution script is a program that configures the benchmark controller and is a program for performing comprehensive benchmarks that is specialized for COSBench. The benchmark execution script uses the program cli.sh that executes the load test installed on the controller of COSBench. Benchmark can be performed by giving an XML file name that contains information necessary for the workload as an argument when executing this program. Since benchmarking is performed repeatedly changing the workload, the benchmark execution script generates an XML file describing all the workloads in advance.
Workloads executed with cli.sh are given a workload identifier, and the benchmark output of different workloads can be distinguished by this identifier. The program of the visualizer described in Section III-B2 also has a function to save this identifier as data in order to align the data using this identifier.

4) Data formatting script
The data shaping script is one of the programs that make up the visualizer and is a program that organizes the data output by the benchmark execution script. The bandwidth information to be evaluated is extracted from the data of multiple workloads output by the benchmark execution script, sorted, and compared with the steady-state benchmark result performed before the fault tolerance evaluation is performed, Normalize the implemented daily steady-state benchmark and output it. The output data is shaped as a format that can be read by gnuplot used in the visualization script in a next Section.

5) Visualization script
The visualization script is one of the programs that compose the visualizer, and the data output from the data shaping script is output using gnuplot. gnuplot is graph utility software based on command line operation that runs on various operating systems such as various UNIX operating systems and Windows. The visualization script can change the width of the x-axis and the y-axis according to the arguments given and is an implementation that enables adaptive visualization according to the range of the benchmark.

V. Evaluations

Authors evaluate the disaster tolerance of CLOUDIAN HYPERSTORE quantitatively using CLOUDIAN HYPERSTORE which is constructed in Section IV-A3 and the SDN-FIT system implemented in Section IV-C. Then the authors visualize the results of evaluations. The network consisting of three sites constructed in Section IV-B is connected by three independent connection lines, and the failure pattern of a single failure on the network is the following three patterns.

1. Osaka-Tohoku
2. Ryukyu-Osaka
3. Tohoku-Ryukyu

When a double failure occurs, the connectivity of the two sites is maintained but the split-brain state is isolated from the other one. As described in Section IV-A-3, the CLOUDIAN HYPERSTORE bucket used in this evaluation has the policy to return an ACK when all 4 sites have been replicated in all 3 sites. Unable to complete the PUT process, all COSBench workloads fail. Because triple failure also causes all workloads to fail for the same reason, this evaluation does not evaluate double failure and triple failure.
In this evaluation, the minimum value of the file size is set to 64 KB by using benchmark_exec command, and the evaluation is performed with nine file sizes up to 16 MB by doubling each time. Also, the workload read:write (RW) ratio is increased by 10 % from 0 % to 100 %. After each failure pattern is implemented, this workload is executed, and the non-steady-state benchmark results are normalized using the previously measured steady-state benchmark results (Figure 5). The horizontal axis is the logarithm axis of the file size, and the vertical axis is the RW ratio, which is visualized as a three-dimensional color map. This can be visualized as a

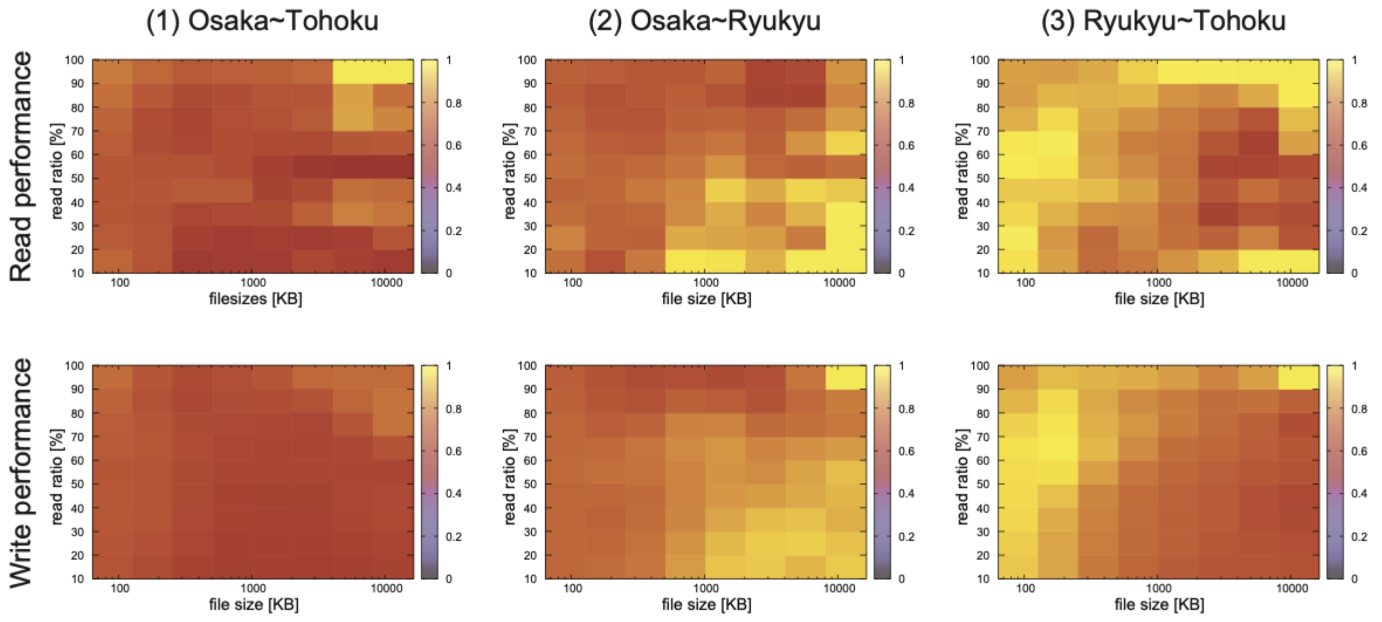


Figure 6. Normalized averaged R/W performance

two-dimensional heat map by displaying it in the gaze direction of a vector parallel to the Z axis. With this heat map, it is possible to grasp the relative quality deterioration in the unsteady state for each failure pattern.

As described in Section IV-C, when you deactivate the VyOS interface connected to the backbone network connecting each location, OSPF detects it and re-routes the routing table in the topology where one of the backbone networks is lost. Calculate and send Link State Update packets to VyOS at other sites, and the routing control tables of VyOS at all sites are updated. In the steady state, all locations can reach all other locations with one hop.

Among them, the one with the largest increase rate of the delay time is at the time of failure occurrence between Osaka University and Tohoku University of failure pattern 1, and the delay time is 14.81 ms at steady state, and it is 56.71ms which is 3.8 times. The average transfer rate of normalized Read and Write at the failure pattern is shown on the left side of Figure 6. The average transfer rate of Read is 8MB/s, and the RW ratio shows the worst value of 0.42 at 60 %, which indicates that only 42 % performance can be obtained compared to the average transfer rate at steady state. Moreover, the average transfer rate of Write is less than 0.83 in the whole area, and the worst value is 0.56 in 10% of RW ratio of 4 MB.

The normalized average transfer rate of Read and Write at failure pattern 2 (Osaka-Ryukyu) is shown in the center part of Fig. 2. In case of the failure pattern, the delay time of Ryukyu University-Osaka University is 23.56 ms at steady state, which is 2.0 times that of 48.55 ms. It is clear that the tendency of performance degradation at failure 2 is different from the tendency of performance degradation at failure 1 and both Read and Write, and the file size is large, and the smaller the RW ratio, the lower the performance deterioration. This means that if the file size tends to be large and the RW ratio tends to be small in the use of wide area distributed services, CLOUDIAN HYPERSTORE in this verification environment can be said to have high fault tolerance, but conversely, the use of small files is large, and authors show that fault tolerance is low when the RW ratio tends to be high.

The normalized average transfer rate of Read and Write at failure pattern 3 (Ryukyu-Tohoku) is shown on the left side of Fig. 3. In case of the failure pattern, the delay time of Tohoku University-Ryukyu University, which is 35.11 ms at steady state time, is 37.33 ms which is 1.1 times. The tendency of performance degradation at failure 3 differs from the tendency of performance degradation at failures 1 and 2 and both Read and Write, and it can be seen that the larger the file size and the smaller the RW ratio, the larger the performance degradation.

These results are the non-stationary average transfer rates obtained by changing the file size and the read: write ratio normalized with the steady-state values. In addition to the fact that the read: write ratio differs for each wide-area distributed service, the ratio changes with the time zone even for the same wide-area service. Similarly, the same is true for file size. Therefore, the service provider can quantitatively grasp the deterioration of the service quality at the time of non-station according to the state of provision of the own service.

Looking at failure pattern 1, the proportion of colors close to 1, ie, yellowish colors, is smaller than the average transfer rate at the time of failure occurrence of the other two patterns. As can be seen from Table 1, the rate of change of RTT is 1.9 to 3.6 times that of the other patterns in case of the failure occurrence pattern of 1 compared to the steady state. It can be said that the output heat map represents the effect.

A. Evaluation of time and accuracy required for failure injection

As described in Chapter 1, the total number of failure patterns increases exponentially as the number of networks connecting each site number increases. When FIT is performed manually, the time required to implement a fault on a router is sufficiently small time for fault tolerance verification as a whole, but due to the increase in the number of sites and the

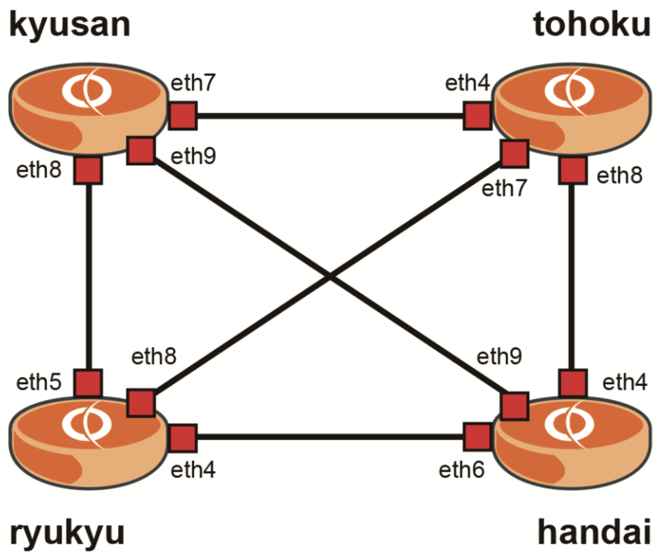


Figure 7. The assumed environment used to evaluate failure injection time and accuracy

number of networks, the time required for manual input is also possible. It also increases exponentially. In addition, it is possible to make an incorrect input by manual input.
In this section, authors measure the time required for manual input, determine whether the NOS command of the input router is correct, and calculate the wrong answer rate. The network used for evaluation prepared the topology connected in mesh form with 4 nodes and 6 networks (Figure 7).
This topology diagram describes the identifiers of the NICs at both ends of the network connected to other sites at each site and provides the subject with an example of command input in the case of causing single failure or double failure as a manual. An example of command input for single failure and double failure is shown below.

```
$ ssh kyusan
$ configure
$ set interface ethernet eth8 disable
$ commit
$ save
$ exit

$ ssh ryukyu
$ configure
$ set interface ethernet eth5 disable
$ set interface ethernet eth8 disable
$ commit
$ save
$ exit
```

The subjects were four persons aged 22 to 42 with the background information system. The subjects were asked to display 22 types of failure patterns from single failure to triple failure, and each failure pattern was input in an environment simulating the console of NOS. The time required for the input was measured, and the correctness of the input result was judged.
Figure 9 shows a graph comparing the time taken to implement the fault by the proposed method and manually. Since the FIT controller implements the fault in the router

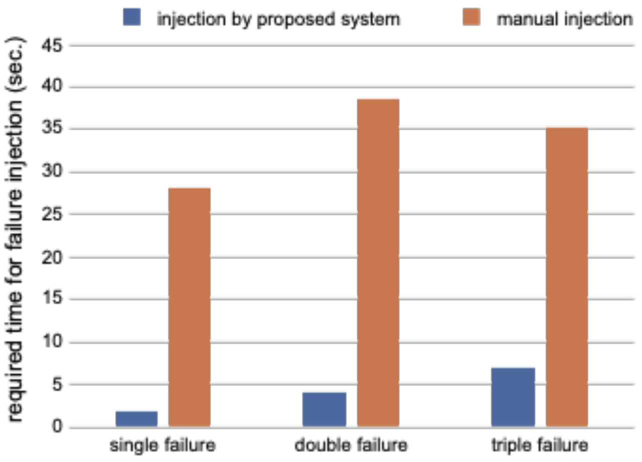


Figure 8. Required time for failure injection by the system and manually

according to the fault pattern output by the fault pattern generator, the time required for a single fault is 1.854 seconds on average. Since the current FIT controller implementation expands multiple faults into multiple single faults and implements faults, the time required is linearly increased while the number of double faults, triple faults, and fault points increases. Yes. The time required to execute a commit command and the save command required to complete a session is dominant in the time required to implement a single failure in a router. Therefore, it is thought that the time required by the proposed method can be reduced by implementing the implementation of the FIT controller in case of causing failures in multiple NICs at one site in one session.
On the other hand, it can be seen that the time required to implement a manual failure is an average of 28.17 seconds for a single failure, which is 15.2 times longer than that of the proposed system. It takes an average of 38.48 seconds for double failure, but 35.22 seconds for triple failure, which is not a simple increase. This is considered to be due to the fact that the failure pattern of the triple failure in the 22 types of failure patterns listed can complete the execution of the command with one router.
The error rate of manual fault implementation was 25% for single fault, 5.6% for double fault and 6.3% for triple fault. The reason for the high error rate in a single failure is thought to be that the single failure was concentrated in the early stages of 22 different failure patterns, and it took time to establish the understanding of the provided manual. Although triple failure is a failure pattern that can be completed by one router command as described above, the error rate is higher in triple failure than double failure. From this, it can be considered that the error rate increases as the number of input command increases.
One of the problems in manual fault implementation is that the worker is forced to wait until the end of the benchmark after the fault implementation. Although the constraint time required to implement a fault is on the order of seconds, the constraint time from fault implementation to fault release is on the order of time, which makes manual fault tolerance verification difficult as the scale increases. ing. The error rate

of the implementation of the fault by the proposed method is 0% in all faults.

VI. 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.

In order to evaluate the effectiveness of this proposal, 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.
In order to evaluate the cost reduction by automation of the fault tolerance evaluation of this proposal, the time required for failure occurrence was measured and compared between the proposed method and the manual case. In single failure, double failure, and triple failure, it was confirmed that the proposed system finished processing in less than 20% of the time required for manual failure implementation. It is also found that the probability of performing incorrect fault implementation for a given fault pattern occurs with a probability of 5% or more in the manual case. From this, it is shown that the proposed system reduces human restraint time and realizes accurate fault implementation.

Acknowledgment

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. Thanks for Cloudian Inc. They always provided technical support of CLOUDIAN HYPERSTORE on the Distcloud environment.

References

[1] I. Nakagawa, K. Ichikawa, T. Kondo, Y. Kitaguchi, H. Kashiwazaki, and S. Shimojo, “Transpacific live migration with wide area distributed storage,” in 2014 IEEE 38th Annual Computer Software and Applications Conference, July 2014, pp. 486–492.
[2] I. Nakagawa, H. Kashiwazaki, S. Shimojo, K. Ichikawa, T. Kondo, Y. Kitaguchi, Y. Kikuchi, S. Yokoyama, and S. Abe, “A design and implementation of global distributed posix file system on the top of multiple independent cloud services,” in 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), July 2016, pp. 867–872.
[3] Y. Rekhter and K. Kompella, “Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling,” RFC 4761, Jan. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4761.txt
[4] A. Viswanathan, E. C. Rosen, and R. Callon, “Multiprotocol Label Switching Architecture,” RFC 3031, Jan. 2001. [Online]. Available: https://rfc-editor.org/rfc/rfc3031.txt
[5] Q. Zheng, H. Chen, Y. Wang, J. Zhang, and J. Duan, “Cosbench: Cloud object storage benchmark,” in Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, ser. ICPE ’13. New York, NY, USA: ACM, 2013, pp. 199–210. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479900
[6] D. Ferguson, A. Lindem, and J. Moy, “OSPF for IPv6,” RFC 5340, Jul. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5340.txt
[7] F. L. Faucheur, P. Merckx, T. Telkamp, R. Uppili, and A. Vedrenne, “Use of Interior Gateway Protocol (IGP) Metric as a second MPLS Traffic Engineering (TE) Metric,” RFC 3785, May 2004. [Online]. Available: https://rfc-editor.org/rfc/rfc3785.txt