Technical Requirements of Experiment Support Software for Huge-Scale Network Environment

Shingo Yasuda, Kunio Akashi, Masatoshi Enomoto, Shinsuke Miwa, and Yoichi Shinoda

***Abstract*—**Network testbed has developed experiment software which supports operations of testbed experiments. In recent years, the experimenters utilize virtualization technologies such as KVM and Xen to expand the scale of experimental environment. By contrast, current experiment support software does not have enough functions for such an experiment environment. As a result, the experimenters often encounter some sort of stumbling block when constructing huge-scale experiment environment. In this paper, we investigate the technical requirements of experiment support software for huge-scale network environment by analyses and considerations.

***Keywords*—**Network testbed, Experiment support software, Large data distribution, Reliable multicast, Virtualization, Resource estimation

I. Introduction

THE Internet and cloud computing services are growing increasingly popular. At the same time, user applications of decentralized architecture such as peer-to-peer (P2P), have also grown in popularity. The technologies underlying those applications and services construct large-scale distributed systems, which have complex mechanisms. Developers therefore must verify not only simulation evaluation but also emulation evaluation using actual program code on target scale network environments. Thus, the importance of network testbeds for verifying such technologies and applications is growing, and the scalability requirements of network testbeds are increasing beyond those currently used.

The StarBED Project[1] has developed experiment support software (hereinafter called the "ESS") called SpringOS, which supports operations of testbed experiments. SpringOS has achieved some positive results in conducting experiments, though we often encounter some sort of stumbling block when constructing an experiment environment on network testbed. Basically, the scale of experiment environment is limited by number of physical nodes which the experimenters utilize. Thus, the experimenters utilize virturalization technology for expanding the scalability of their experiment. However, it increases the service requests from experiment nodes to management servers. One-sided approach for expanding the scale of experiment nodes triggers failure. As a result, it decreases the ESS's reliability and expands the cost for experiment.

The experimenter takes advantage of virtual machines in network testbed to expand the scale of experiment environment. Expanding the scalability of experimental environments using virtual machines requires multiplexing much more virtual machines by allocating resources onto each virtual machine appropriately. Generally, the roles of several experiment nodes are different. Consequently, it will be excess or deficiency of resources if the experimenter evenly allocates resources to all virtual machines. Hence we need mechanism to allocate and provision the computer resources such as memory, CPU and network traffic if the experimenter utilizes virtual machine. Current ESS does not have enough function such as failure avoidance, failure recovery and resource allocation for huge-scale experiment environment. As a result, it is difficult to support experimenter's operations.

In this paper, we investigate the technical requirements of experiment support software for huge-scale network environment by analysis and considerations.

II. Related Works

A. Typical Testbeds

In research and development area, various testbeds such as Emulab[2],[3], PlanetLab[4],[5], and StarBED[1] are used to validate network applications or technologies over an experiment environment. The major problem is cost in constructing experiment environments on a large number of nodes and network equipments. Moreover, cost is also needed to maintain testbed equipments.

Most of these testbeds have supporting software that helps in constructing the target experiment environment and conducting network experiments on it. PlanetLab has supporting software such as PlanetLab Central (PLC) and CoDeploy[6], while StarBED has SpringOS[7],[8],[9]. SpringOS is a particularly compelling software suite consisting of a dozen modules, each with special roles.

S.Yasuda is with the Japan Advanced Institute of Science and Technology , 1-1 Asahidai Nomi, Ishikawa, Japan (corresponding author to provide phone: +81-761-51-1699(1327); e-mail: s-yasuda@jaist.ac.jp).

K.Akashi is with the Japan Advanced Institute of Science and Technology , 1-1 Asahidai Nomi, Ishikawa, Japan (e-mail: k_akashi@jaist.ac.jp).

M.Enomoto is with the Nara Institute of Science and Technology , 8916-5 Takayama Ikoma, Nara, Japan (e-mail: masatoshi-e@is.naist.jp).

S.Miwa is with the National Institute of Information and Communications Technology , 2-12 Asahidai Nomi, Ishikawa, Japan (e-mail: danna@nict.go.jp).

Y.Shinoda is with the Japan Advanced Institute of Science and Technology , 1-1 Asahidai Nomi, Ishikawa, Japan (e-mail: k_akashi@jaist.ac.jp).

B. StarBED and SpringOS

StarBED is a testbed which includes about 1000PCs and many switches, and the all nodes are interconnected. All StarBED PC nodes have a sort of Remote Management Interface(hereinafter called "RMI") such as Wake on LAN (WoL)[10], Intelligent Platform Management Interface(IPMI)[11] and HP Integrated Lights-Out(ILO)[12]. StarBED provides various services which manage facilities and basic service softwares such as DHCP, DNS, NTP, File Server and external(internet) access. Moreover, in the StarBED project, SpringOS is developed for dozen functions such as resource management, network configuration and driving the experiment based on scenario. Many experimenters can construct the targeted experiment environment by using SpringOS in the StarBED, which is available to conduct the experiment.

Many research projects conducted experiments using a few hundred of physical nodes in recent years in StarBED. In addition, some research projects conducted experiments with over 10,000 virtual nodes in StarBED using virtualization technology[15].

C. Construction Procedure Using SpringOS

In order to construct the experimental environment, the experimenter needs to install OS or application on physical nodes of StarBED, distribute the data such as configuration file or virtual node OS image, and configure the network topology. There is common procedure to construct an experimental environment what we will describe in this section. In addition, it becomes popular to construct an experimental environment using virtualization technologies in recent years. Thus this section describes construction procedure using virtualization technologies on StarBED.

1) Software Installation on the Nodes

In order to conduct experiments, the experimenter needs to install OS and application software on physical nodes. There are two methods for it.

a) The experimenter installs OS and application software on one physical node. This process is making a template diskimage of the experimental node. After that, experimenter distributes that template diskimage to other physical nodes. Here, SpringOS has function, which distribute that template to other nodes.

b) The experimenter creates network boot OS image and installs OS and application on one physical node. After that, it is booted by network boot. Network boot means diskless boot by network services. Most poplar technology is Preboot Execution Environment(PXE)[13] Boot. All of StarBED physical nodes have network interface card which includes function of PXE Boot.

2) Network Configuration

The experimenter needs target network topology for the experiment. In StarBED, the experimenter can configure the network topology by VLAN without changing the physical topology. In addition, SpringOS can configures VLAN settings of switches for the experiment environment.

3) Scenario Driving

After the construction of experiment environment, experimenter can experiment on the environment. SpringOS has function of experiment execution which utilizes scenario. Scenario consists of command lists and timing triggers. SpringOS distributes scenarios to each experiment node, and a scenario execution program running on the experiment nodes executes the commands in the scenario. This mechanism can also synchronized scenario execution which utilizes passing messages.

D. Large Data Distribution

One of major roles of ESS is distributing certain OS images and installing them on each node. SpringOS, for instance, uses FTP to transfer the data. SpringOS version 1.5 is able to distribute an 870 MB disk image among 230 nodes in about 2,200 seconds [14]. The elapsed time for 50 nodes is 1,000 seconds and it increases if the number of nodes increases. That paper [14] describes the reason as "FTP server-side problem, including NIC's capacity or HDD reading speed". Many research projects in recent years have conducted experiments using a few hundred physical nodes in StarBED. Some research projects have also conducted experiments with over 10,000 virtual nodes in StarBED using virtualization technologies such as KVM and Xen [15]. The disk image distribution elapsed time therefore significantly affects construction of the experiment environment.

Frisbee [16] designed and evaluated a scalable disk distribution system. They conducted an experiment to distribute an OS disk image to 80 nodes with reliable multicast for evaluating system writability. For that experiment environment, the sender server interconnected with client nodes via 100 Mbps Ethernet. Gigabit Ethernet has become popular in recent years. StarBED thus also utilizes it for all experiment nodes and utilizes 10 Gigabit Ethernet in some parts of the management network. The writing speed of the common computer's hard disk is slower than Gigabit Ethernet, so disk writing speed can lead to a bottle-neck in large data distribution. Common computers, operating systems and applications have a buffer for disk writing, but its effectiveness is limited by memory size, so we need to validate the effect of the buffer on large data distribution.

III. Categorizing the Errors

This section describes issues of ESS's operation failure when constructing the experimental environment on StarBED. The experimenter encounters various troubles while constructing the environment. At first, we describe major troubles when constructing the experimental environment on StarBED. Figure 1 shows the number of successfully booted nodes in startup nodes using PXE boot in StarBED. Table 2 lists experimental condition, and Table 1 lists experiment node specifications.

This is a basic performance in StarBED facilities because this experiment does not use any failure recovery mechanism and operation except SpringOS's mechanism. There are few nodes which can't startup normally in the case of over 50

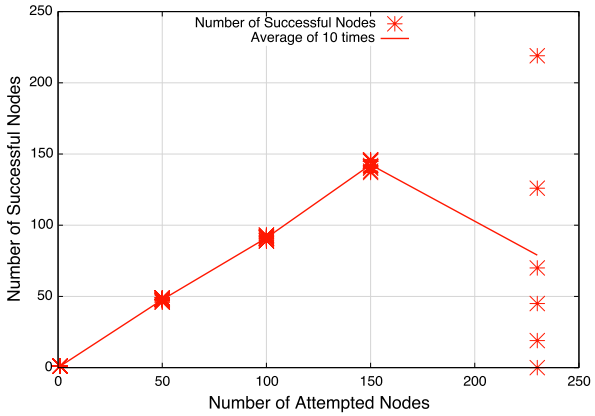

Fig. 1 Number of Successfully booted nodes

nodes. Furthermore, in the case of 230 nodes, the result describes the success rate in wide range.

There are three reasons to get such result: (1) lacking reliabilities of RMI, (2) management service failure such as collision of path to management server (3) network service overload when starting physical nodes.

A. *Reliability of RMI*

The RMI problem appears when starting physical nodes. In most cases, there are two problems; (1) no response, (2) difference between response from RMI and actual statement of RMI. Moreover, there is a case that sender disconnected from RMI abnormally when RMI with type of connection oriented communication. It is also categorized as RMI problem. In this case it is likely caused by packet loss or a certain kind of bugs happened in hardware overload. Usually it is possible to recover by retrying the same process.

By contrast, even though network is not overloaded, there is also the case that error is occurred. Moreover, the case of difference between response from RMI and actual statement of RMI occurs when the experimenter or ESS sends many commands in a short period. These problems are likely caused by design failure of RMI hardware. This phenomenon occurs in three products of different manufacturers that we are using. We define this phenomenon as synchronization problem between software and hardware in RMI product. It seems that RMI product needs an interval to change the hardware state after receiving command. When the experimenter executes next command, the above phenomenon does not appear by setting the intervals for 10 seconds or more since last command.

B. *Problems of Management Server*

The problems of management server are experimental management server errors that nodes can't be serviced such as DHCP and NFS. It is a matter of a service software error, which is not a hardware error. When the experimenter controls the experimental nodes, the above phenomenon does not appear by controlling the intervals or distributing service requests to the servers.

Hence, We proposed and implemented a mechanism called mount point redirection, as shown in Figure 2. All experiment nodes are installed on the NFS server in this mechanism. In addition to insert a few seconds as an interval between each

Table. 1 Node Specifications

Server	CPU	Memory	Disk	OS
TFTP/ DHCP	Xeon E5405	2GB	-	Solaris 10
NFS	Xeon E5405	16 GB	-	Solaris 10
Group H	Xeon X3350	8 GB	SATA 160 GB	-

Table. 2 Experiment Conditions

Node Group	Group H
No. of Nodes	1, 50, 100, 150, 230
Boot Type	PXE Boot
OS	Debian Linux Custom
OS Image Size	193MB
No. of NFS Servers	1
No. of TFTP &DHCP Servers	1 (2 in 1)

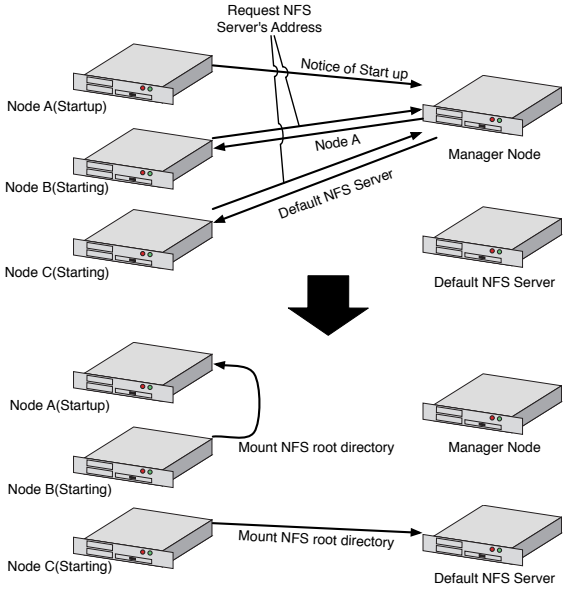

Fig. 2 Mount Point Redirection

booting command. Due to this, there are nodes that already have same data which are required. When boot experiment nodes, manager node redirect NFS server address which already have requested data to request node. When many nodes request same data, this mechanism makes NFS server load distribution.

C. *Problems of Hardware*

The problems of hardware are errors occurred when the hardware detects the breakdown such as CPU error, power supply trouble and disk trouble. These errors can be tentative or not. The experimenter can recover the tentative error by restarting the system. The experimenter needs to judge if it is

the tentative error or not by trying to restart the system. If it is not tentative error, the experimenter decides to exclude that node from experimental nodes. It can decrease the number of physical nodes for the experiment. Therefore the experimenter needs to construct a certain numbers of spare nodes in advance in order to replace broken nodes with them.

IV. RMI Failure Recovery

In section III.A, we describe reliability of RMI. Current RMIs do not have enough reliability in large-scale network testbed because it utilizes UDP packet for communication and so on. Nevertheless RMI is most basic and important feature for ESS. Hence we should enhance the reliability of RMI control.

A. RMI Command Procedures

When controlling the RMI, the experimenter needs to consider the following points as described in section III.A.

- Retry when the connection terminates abnormally.
- Set enough time for message interval.
- Confirm node's statement to same as operation command.

We propose simple mechanisms based on these conditions. Figure 3 shows state transition diagram of operation "**Get Power Status**". It retransmits when sender gets no response or connection error occurs. However, it resends the message until three times because the reactivation process will not end if trouble is not temporal. This diagram will be same for "**Power Reset**". Moreover, manager server manages the electrical power control and start confirmation of a physical node. Next, in order to confirm if operation works as per command, the procedure to check state variation of power supply in physical is necessary. Figure 4 shows state transition diagram "**Set Power On**". At first, it confirms the power state of target node. If the power statement is off, it sends command of "**Power On**". After send command, it reconfirms the power state of target node, and resend command of "**Power On**" again if the power state is off. But, it resends until three times too due to same reason of "**Get Power Statement**". Since the operation of changing power state is same in both "**Set Power On**" and "**Set Power Off**", the diagrams will be the same.

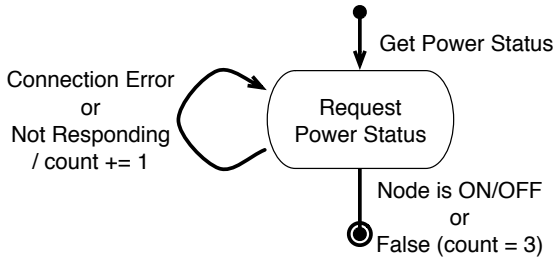

Fig. 3 State transition diagram of command sender for getting power status

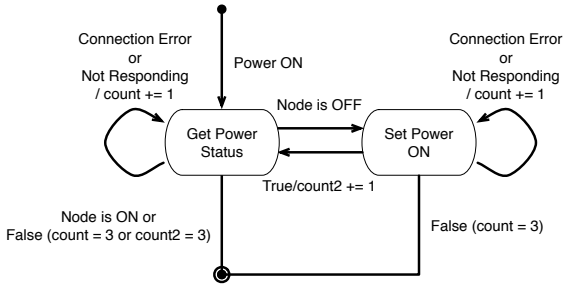

Fig. 4 State transition diagram of command sender for changing power to on

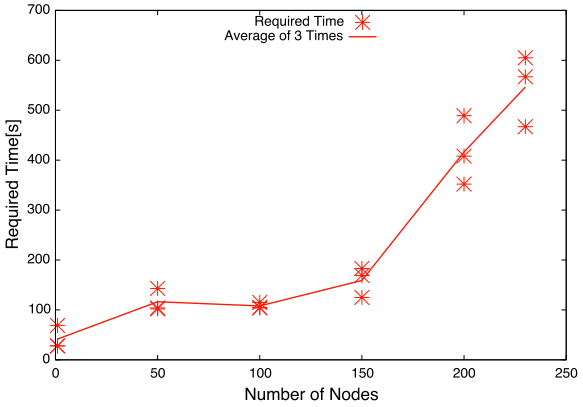

Fig. 5 Required Time for Distribution

V. Multicast Data Distribution

In case of the experimenter utilize virtualization technology in his experiment. The physical nodes are virtual machine host nodes, and the experimenter must distribute a virtual node disk image to these nodes. If all virtual machine host nodes have booted normally, the NFS service experiences a bottleneck in distribute large amounts of data such as virtual node disk images to a large number of physical nodes, so we proposed using reliable multicast for large amounts of data distribution in the testbed. Thus we conducted data transfer experiments. In this work, we used "Makuosan", a software "Multicast All-Kinds of Updating Operation for Servers on Administered Network"[19], in the experiment. This supports reliability by NACK-Based retransmission. Figure 5 shows the result of the experiments, Table 3 lists experiment conditions, and Table 1 lists experiment node specifications. The NACK-based retransmission method has a problem of NACK implosion, so the required time becomes longer while the number of receiver nodes increases. The sender node receives a burst retransmission request from receiver nodes in this experiment. This is a serious problem and we are interested in the reasons

Table. 3 Experiment Conditions

Node Group	Group H
No. of Sender Nodes	1
No. of Receiver Nodes	1, 50, 100, 150, 230
Distribute Data Size	2.7 Gbytes
Experiment Node's OS	Debian Linux
Makuosan Version	1.3.1

behind it.

A. Analysis of the Bottleneck

Packet loss usually occurs at a low rate in a high-performance and closed network such as StarBED. As mentioned above, the sender node often receives burst retransmission requests from receiver nodes. Accordingly, there are some bottlenecks in this type of network environment. We therefore conducted some experiments to analyze the reasons for this.

1) Reasons for Retransmission Occurrence

Packets are obviously dropped on the path between sender and receiver due to a burst retransmission request, so we experimented on analyzing the packet loss point on the path between sender and receiver nodes, as shown in Figure 6. This shows the flow of IP multicast data transfer. The sender node interconnects with the network switch by 1 Gb Ethernet, and 1 Gb Ethernet integrates the switch with each receiver node. Packets are sent from sender nodes to receiver nodes via link local multicast. The total transfer size is 100 MB. There are 90 receiver nodes and only one sender node. Table 3 lists the experiment node specifications. A sent packet includes: field of data length (4 bytes), field of packet sequence number (4 bytes), and variable field of data. The length of the data field is written in the field of data length, and its longest length is 1400 bytes. The sender program is written in Ruby. The sender program has high data performance of about 800 Mbps.

The measurement points include the sender node's upstream mirror port and the receive node's interface. The tcpdump program captures data. As results we found that packet loss is not observed at the mirror port. The switch did not drop the packets for packet switching at over 800 Mbps of throughput utilizing unicast UDP packets.

Figure 7 shows the experiment's results. The Y 1 axis shows the received packet sequence number, and the Y 2 axis shows the packet loss rate. The X axis shows time transition. We do not show the packet loss of the sender mirror port in the graph because it is not observed. We confirmed the basic performance of this experiment environment. The sender sent all packets for about 1 second. The receiver nodes received the packets with over a 90% packet loss rate with 0.1 seconds delay until the sending process ends. The sequence number of all packets received by 90 nodes was also the same. This result seems to show the major reason for packet loss in IP multicast. A specification or configuration issue of the network switch can, however, cause this result.

2) HDD Writing Speed

The next reason for the bottleneck is the transfer speed to write to or read the disk if the network can pass the traffic at up to full speed. As shown in Figure 6, the sender node has a local disk (HDD), as do all receiver nodes. The sender program reads transport data from the local disk. The receiver program writes received data to the local disk. The writing speed of an HDD or any other storage medium is generally slower than the reading speed. The receiver program will continue dropping packets if the data receiving speed is faster than the disk writing speed. The disk writing speed also differs due to the disk position of data writing.

Modern computers have a buffer for disk writing due to the buffer writing performance from the slowness of the disk

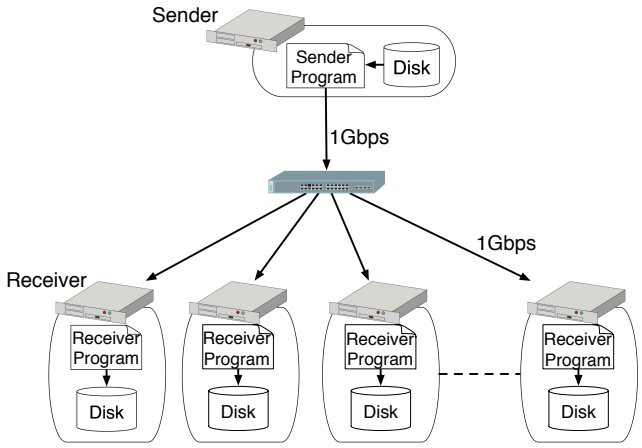

Fig. 6 Flow of Multicast Data Transfer

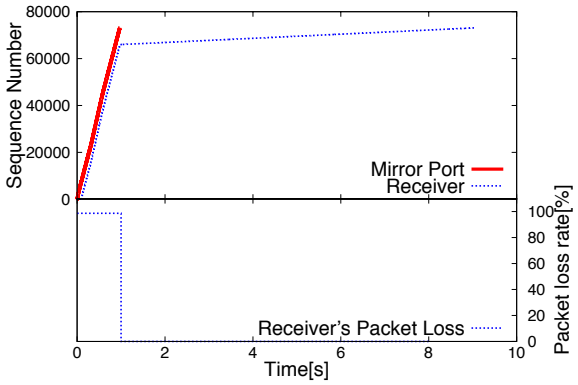

Fig.7 Packet Loss Rate

writing speed. We therefore experimented with two types of scenarios to analyze the effect of the disk writing speed. The first experiment was for analyzing the effects of disk writing speed by the writing buffer and difference of disk writing speed due to the position of data writing. We utilized one node of StarBED Group H in this experiment and using Python wrote a simple program for disk writing. Table 1 shows the node specifications. The program write timestamp is acquired by timeObject.time() with dummy data to the local disk. The data length is 1472 Bytes one time and 8 GB total data size. The next line of fileObject.write() is fileObject.flush() in the program due to analyzing the basic performance of the disk. We divide the hard disk into 10 partitions for changing the writing position. The program writes 10 times in each partition.

Figure 8 shows the experiment results. The Y axis shows the disk writing speed. The X axis shows the time (in seconds). There are peaks of disk writing speed until 15 seconds. This is the effect of a low-layer buffer such as the cache in the HDD.

The second experiment was for writing the received packets to the local disk in order to analyze the effect of the disk writing speed. We utilized two nodes in this experiment as sender and receiver. These are from Group H, as shown in Table 1. The sender program sends 8 GB data by using unicast UDP packets to the receiver. The receiver program writes the received data on the local disk. The sender has a function of flow control for transport speed. We measure written data

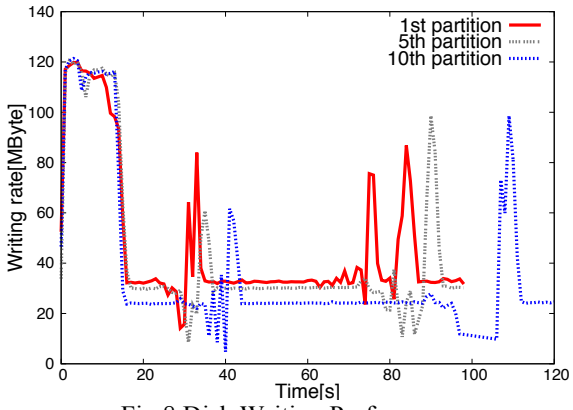

Fig.8 Disk Writing Performance

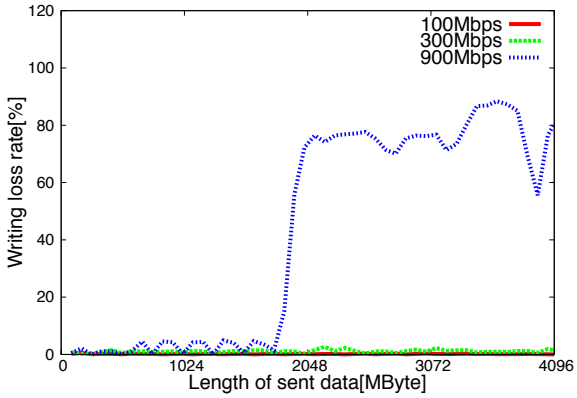

Fig. 9 Average of DDisk Writing Loss Rate

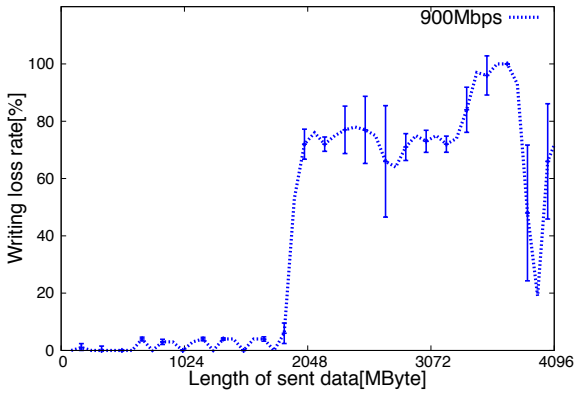

Fig. 10 Disk Writing Loss Rate and SD (900Mbps)

blocks on the receiver's local disk by three bandwidth conditions.

Figure 9 shows the experiment results. Each point represents the average of 10 experiments over an 80 MB window. This figure also only shows 4 GB in front of 8 GB sending data. The Y axis shows the packet writing loss rate. The X axis shows the transfer data position. The loss rate of written data is low and almost unchanged, and even the sender program sends data at 100 Mbps and 300 Mbps. In contrast, there are elbows of writing loss rate in the case of 900 Mbps, so the writing loss rate is over 60% after the elbow.

Figure 10 shows the average and standard deviation of 10 times in the case of 900 Mbps. The writing loss rate is different at each point in time, which suggests that writing loss data block is different in each node.

These two experiments identified the shift of throughput during large data distribution. In addition, these experiments also identified the difference between disk writing loss and packet loss in network switch. Data block of disk writing are different block of each node.

3) Disk and Network Aware Data Distribution

We proposed utilizing reliable multicast so that large data files such as virtual node disk images are distributed to experiment nodes. We describe the requirements of reliable multicast for large data distribution in a testbed. These are as follows:

• Disk speed and network aware congestion control mechanism;

• Multi-network sibling recovery mechanism;

These mechanisms will contribute to optimizing large data distribution in a centralized testbed.

VI. Resource Usage Estimation

In previous sections, we described technical requirements for failure recovery and inhibition. In this section, we discuss ESS functions that is needed for huge-scale experiment environment which utilizes virtualization technology. The experimenters take advantage of virtual machines in network testbed to expand the scale of experiment environments. Expanding the scalability of experiment environments using virtual machines requires multiplexing much more virtual machines by appropriately allocating resources onto each virtual machine.

A. Resource Allocation

In huge-scale experiments, the experimenter need to consider resource allocation including CPU, memory, network resources and so on. Generally, the role of several experiment nodes are different. Consequently, it will be excess or deficiency of resources if the experimenter allocates resources to all virtual machines evenly. Thus the experimenter describes enough quotas of resources into virtual machine's configuration files for each machine. But in a huge-scale experiment environment, the experimenter must describe many configuration files and distribute them to each experiment node.

Hence we proposed mechanism for resource allocation from one configuration file in our previous work[22],[23]. In this implementation, the experimenter can describe resource information of each virtual machine by XML format. On the other hand, the experimenter must decide its quotas of resource allocation for each virtual machine by own-experience.

B. Roles of ESS

There are researches for resource provisioning to allocate the optimum resources without experimenter-experience. Deng[24] proposed estimation approach for bgpd by equation-of-state model of BGP update message. XENebula[15] proposed measurement based memory resource

allocation method.

Whereas, these approaches depend on several target implementation. Roles of ESS is to support for every experiments; therefore, required features for ESS should only support resource allocation. Hence ESS should have mechanism to describe the resource allocation simply such as our previous work[22],[23]. In addition ESS should also have resource state logger in experiment nodes to support the experimenter's configuration of resource quotas.

VII. Conclusion

This paper cites technical requirement for ESS to support huge-scale experiment environment which utilize virtualization technology. We categorize the errors experimenter encountered. As a result, we describe the needs for reliability of RMI, server load distribution technique and traffic efficiency. We proposed three technical requirements for these problems: (1) implementation of NFS server mount point redirection for NFS server load distribution, (2) simple state transition diagram for reliable communication with RMI and, (3) disk-speed and network aware large data distribution mechanism which utilizes reliable multicast for traffic efficiency.

In addition, we cite the issues of resource usage estimation for virtual machine, and describe the need for simple description mechanism of resource allocation. Furthermore we also describe the need for resource state logger in experiment nodes to support experimenter's configuration of resource quotas.

These mechanisms and functions of resource allocation will contribute to optimize reliable construct of huge-scale experiment environment and conduct their experiments in centralized testbed.

Acknowledgment

The authors thank Assistant Professor H.Hazeyama from Nara Institute of Science and Technology for his insightful comments and suggestions. The authors also thank Assistant Professor D.Miyamoto from the University of Tokyo for his insightful comments and suggestions. A part of this research was received generous support from T.Inoue, Ph.D. T.Miyachi, Ph.D. K.Chinen, H.Nakai, M.I.Tariq, Ph.D. LATT Khin Thida and Ph.D. Y.Takano.

References

[1] StarBED Project
http://www.starbed.org

[2] Emulab
http://www.emulab.net

[3] B. White, J.Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experiment environment for distributed systems and networks, pp 255-270. USENIXASSOC, Dec. 2002.

[4] Planet-Lab
http://www.planet-lab.org

[5] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet, In HotNets-I '02, Oct. 2002

[6] A Scalable Deployment Service for PlanetLab
http://codeen.cs.princeton.edu/codeploy/

[7] T. Miyachi, K. Chinen, and Y. Shinoda. Automatic Configuration and Execution of Internet Experiments on an Actual Node-based Testbed. In International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities(Tridentcom 2005), Feb. 2005.

[8] K. Chinen, T. Miyachi, and Y. Shinoda. A Rendezvous in Network Experiment - Case Study of Kuroyuri. In International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities(Tridentcom 2006), Mar. 2006.

[9] T.Miyachi, K. Chinen, and Y. Shinoda. StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In International Conference on Performance Evaluation Methodlogies and Tools(Valuetools 2006), Oct. 2006.

[10] Magic Packet Technology White Paper
http://support.amd.com/us/EmbeddedTechDocs/20213.pdf

[11] Intelligent Platform Management Interface
http://www.intel.com/design/servers/ipmi/

[12] iLO(Integrated Lights-Out 2)
http://h18004.www1.hp.com/products/servers/management/remotemgmt.html

[13] Preboot Execution Environment (PXE) Specification
http://www.intel.com/design/archives/wfm/downloads/pxespec.htm

[14] T. Miyachi, T. Nakagawa, K. Chinen, S. Miwa and Y. Shinoda, StarBED and SpringOS Architectures and their Performance, Tridentcom 2011, Apr, 2011.

[15] S. Miwa, M. Suzuki,H, Hazeyama, S, Uda, T, Miyachi, Y, Kadobayashi and Y. Shinoda, Experiences in emulating 10K AS topology with massive VM multiplexing, Proceedings of The First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures(VISA 2009), Aug, 2009.

[16] M.Hibler, L.Stoller, J.Lepreau, R.Ricci and C.Barb. Fast, Scalable Disk Imaging with Frisbee, In Proc. of the 2003 USENIX Annual Technical Conference, pp.283-296, Jun, 2003.

[17] A. Mankin, A. Romanow, S. Bradner and V. Paxson, IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols, RFC2357, Jun, 1998.

[18] B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd and M. Luby, Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer, RFC3048, Jan, 2001.

[19] Makuosan(Multicasts All-Kinds of Updating Operation for Servers on Administered Network)
http://lab.klab.org/wiki/Makuosan (in Japanese)

[20] B. Adamson, C. Bormann, M. Handley and J. Macker, Negative-acknowledgment (NACK)- Oriented Reliable Multicast (NORM) Protocol, RFC3940, Nov, 2004.

[21] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, IEEE/ACM Transactions on Networking, Dec 1997, Volume 5, Number 6, pp. 784- 803.

[22] S. Miwa,T. Miyachi,M. Eto, M. Yoshizumi, Y. Shinoda, Design and Implementation of an Isolated Sandbox with Mimetic Internet used to Analyze Malwares, DETER Community Workshop on Cyber Security Experimentation and Test 2007 (DETER07), 2007.8.

[23] S. Miwa, T. Miyachi, M. Eto, M. Yoshizumi, Y. Shinoda, Design Issues of an Isolated Sandbox used to Analyze Malwares, In proceedings of Second International Workshop on Security(IWSEC2007), LNCS 4752 Advances in Information and Computer Security, ISBN 978-3-540-75650-7, pp.13-27, 2007.10.

[24] W. Deng, P. Zhu, X. Lu, and B. Plattner. On Evaluating BGP Routing Stress Attack. Journal of Communicasions, 5(1):13-22, Jan, 2010.

S. Yasuda (M'11-) is a doctoral student at Japan Advanced Institute of Science and Technology, Japan. His research interests include Network Testbed, Delay Tolerant Networking, Wireless Network Emulation, and E-Learning System. He is working on StarBED Project.

K. Akashi is a doctoral student at Japan Advanced Institute of Science and Technology, Japan. His research interests include Network Testbed, and Network Emulation. He is working on StarBED Project.

M. Enomoto is a doctoral student at Nara Advanced Institute of Science and Technology, Japan. His research interests include Network Testbed, Testbed

Migration, and Computer Resource Allocation. He is working on WIDE Project.

S. Miwa is a director of the Hokuriku StarBED Technology Center in National Institute of Communications Technology. He received his Ph.D. from Japan Advanced Institute of Science and Technology, Japan, in 1999. His research interests include Network Testbed, Network Emulation, and Network Security.

Y. Shinoda is a Professor in Japan Advanced Institute of Science and Technology. He received his Ph.D. from Tokyo Institute of Technology, Japan, in 1989. His research interests include Distributed and Parallel Computing, Networking Systems, Operating Systems, and Information Environment.