A Platform Independent Tool for Evaluating Performance of Computing Equipment for a Computer Laboratory

Takashi Yamanoue
Kagoshima University
Korimoto, Kagoshima 890-0034, Japan
+81-99-285-7187
yayamnoue@cc.kagoshima-u.ac.jp

ABSTRACT
Designing computing equipment for a computer laboratory is not easy. In a class in a computer laboratory, it is not unusual that all students do the same thing simultaneously. Tremendous traffic is on the network and heavy load is on the file server or the web server at that time. In order to design the equipment, it is useful if we can have performance data which is acquired in a real class. And it is more useful if we can compare the performance data of various kind of equipment which is acquired by doing the same operations in a real class. In order to acquire such kind of data, we are developing a benchmark test tool for distributed systems. This tool records real operations by users on computing equipment and it acquires its performance data. The tool can replay the same operations on different computing equipment. The tool can also let every computer doing the same thing simultaneously. In order to compare the performance data of various kind of computing equipment, this tool is in Java. So it is platform independent. We show the structure of the tool and the experiment of the usage of the tool.

Categories and Subject Descriptors
K.6.2 [**Installation Management**]: Benchmarks, Computer selection, Performance and usage management

General Terms
Measurement, Performance, Human Factors, Experimentation, Verification, Design, Reliability

Keywords
Benchmark Test, P2P (Peer to Peer), Distributed Systems, Java

1. INTRODUCTION
Purchasing equipment for computer laboratories or University's information infrastructures is one of the most important jobs for a university's computing service organization. We have to make the requirement specification which satisfies most of the teachers' and students' requests. Then, we have to design the equipment which can satisfy the requirement. These are very hard work and time consuming[1][2][4]. We have to think of money and the requests. We also have to think of "does it really work?" Most of today's equipment for computer laboratories is a distributed system which consists of many computers and a network. It is easy to see the performance of each element of the distributed system. However, it is hard to see the total performance of the distributed system.

In a class in a computer laboratory, it is not unusual that all students do the same thing simultaneously. Tremendous traffic is on the network and heavy load is on the file server or the web server at that time. So we'd like to write such like "the Web server must return the web page to every client at least in 3 sec. when 200 clients access the web page in 0.5 sec." in the requirement specification. And we'd like to know which kind of web server can satisfy the requirement. In order to make the requirement specification, it is useful if we can have performance data which is acquired in a real class. And it is more useful if we can compare the performance data of various kind of equipment which is acquired by doing the same operations in a real class. In order to acquire such kind of data, we are developing a benchmark test tool for distributed systems, which is called *DSR (Distributed System Recorder)*. DSR records real operations by users on computing equipment and it acquires its performance data. DSR can replay the same operations on different computing equipment. DSR also can let every computer doing the same thing simultaneously.

In order to compare the performance data of various kind of computing equipment, DSR is in Java. So it is platform independent. We show the structure of the tool and an experiment of the usage of the tool.

2. DSR, a Benchmark Test Tool for Distributed Systems
DSR (Distributed System Recorder) is a benchmark test tool which records real operations on a distributed system and measures the performance of the distributed system. This tool acquires the performance data of the equipment by replaying the recorded operations. This tool can replay the same operations on different computing equipment. The tool also can let every computer doing the same thing simultaneously. In order to compare the performance data of various kind of computing equipment, this tool is in Java. So it is platform independent.

SIGUCCS'04, October 10–13, 2004, Baltimore Maryland, USA.

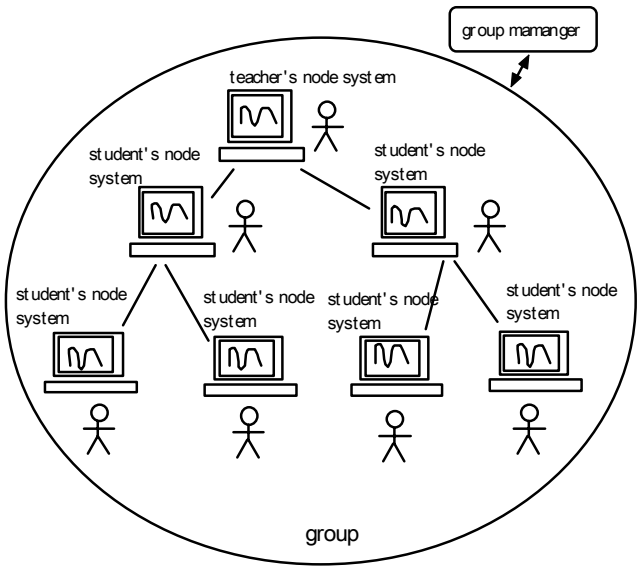

Figure 1. Structure of DSR

2.1 Structure of DSR

DSR consists of three kinds of program. These are the *teacher's node system*, the *student's node system* and the *group manager*. Figure 1 shows the structure of DSR.

2.1.1 Teacher's Node System

The teacher's node system is corresponding to the teacher in a computer laboratory. This node system records the teacher's operation, replay the operation and record the performance data on his/her computer terminal. This node system also can direct the every student's node system to do the same operation as the teacher's, simultaneously. This is realized by broadcasting the commands which are corresponding to the teacher's operation, from the teacher's node system to students' node systems

2.1.2 Student's Node System.

The student's node system is corresponding to a student in a computer laboratory. This node system records the student's operation, replay the operation, and record the performance data on his/her computer terminal. This node system also can interpret and execute the commands which are sent from the teacher's node system.

2.1.3 Group Manager

The group manager manages the group of the teacher's node system and students' node systems. When the teacher's node system broadcasts the commands, every student's node system must receive the commands in short time without errors. In order to realize this, DSR adopts a kind of P2P technology. Every node system in the group has at least one TCP connection to another node system.

If one node system receives a command from one TCP connection, the node system sends the command to another/other node(s) by using TCP connection(s) if the system has the connection(s). The node system interprets and executes the received command after that. The node systems and the connections form a complete binary tree. The teacher's node system is the root node on the binary tree.

If every connection in the group can pass data simultaneously, the time complexity of receiving a command by all students' node systems is $O(\log N)$ where N is the number of the node systems in the group[3]. In order to form the binary tree, the group manager directs a node system where to connect when the node system is becoming a member of the group. Please note that the teacher's node system doesn't need to be the root node.

2.2 Node system

The teacher's node system or a student's node system consists of the *main controller*, *applications*, the *event recorder/player* and the *command transceiver* (Figure 2).

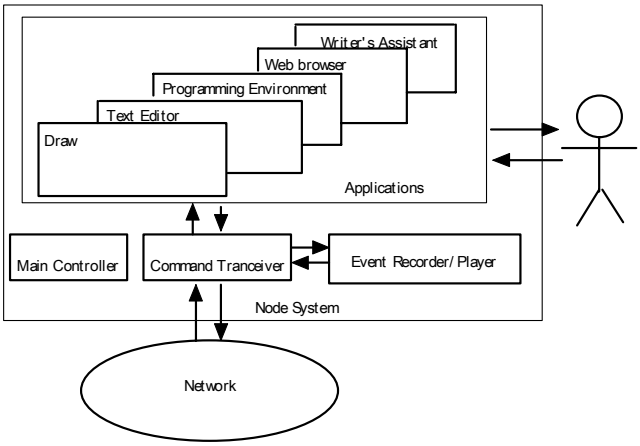

Figure 2. Node system

2.2.1 Main Controller

The main controller is a kind of dispatcher. All applications and the command recorder/player are spawned from this controller. Figure 3 shows the GUI of the main controller.

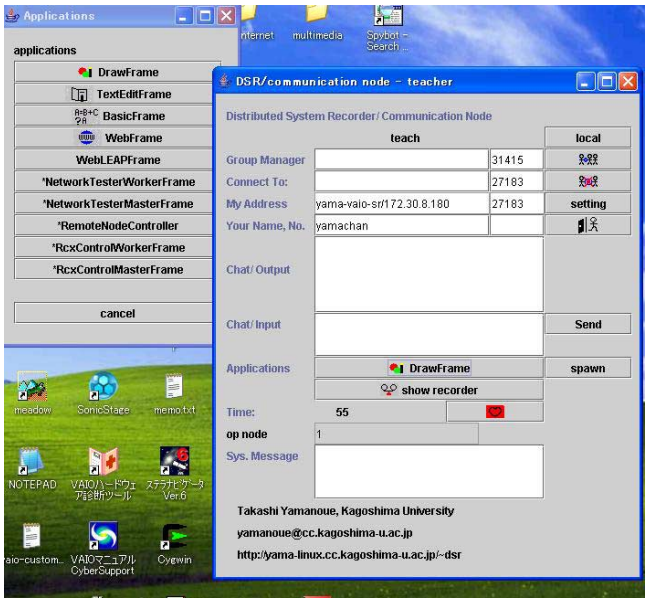

Figure 3. Main Controller

2.2.2 *Applications and Commands*

A node system has applications such as a draw, a text editor, a web browser, a simple programming environment and a writer's assistant[7]. When a user uses an application of the system, commands are generated. These commands are corresponding to the event which is fired by user operation. A command of them is such like "Move the mouse cursor to (x,y) on the draw window" or "Click the button No. x on the programming environment". In order to generate and interpret such kind of commands, An application of the DSR is made of *controlled GUI parts*. Each of these parts generates commands when the user operates the part. Each of these parts is *controlled* by interpreting the corresponding command (Figure 4).

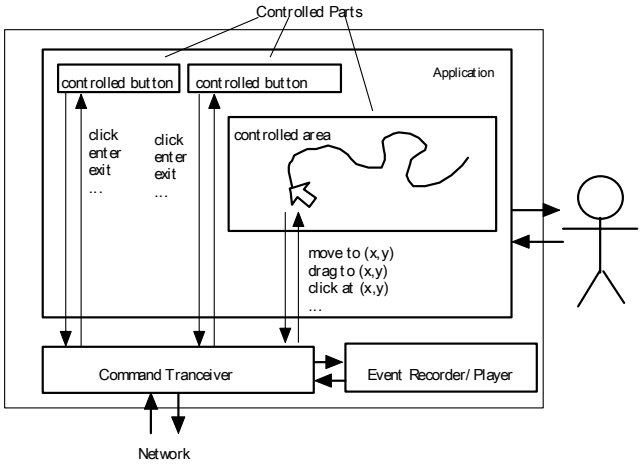

Figure 4. Controlled parts

2.2.3 *Event Recorder/Player*

The event recorder/player can record commands, which are received from another node system such like the teacher's node system, or commands which are generated by it self, with the recorded time (Figure 5).

In order to record the time, the event recorder/player has a timer. The time of this timer is synchronized with timers of other node systems in the group. This synchronization is realized by exchanging messages between nodes in some interval.

A node system can replay the operation by interpreting the recorded commands. The command recorder/player also records messages of applications. The message includes information such like the start/end time of loading the application, start/end time of loading html file from a Web server and start/end time of saving a file to a file server. These messages are in CSV format. We can analyze the recorded message easily by using software such like the Excel.

2.2.4 *Command Transceiver*

The command transceiver transmits and receives the commands between other node systems. The command transceiver also interprets the commands.

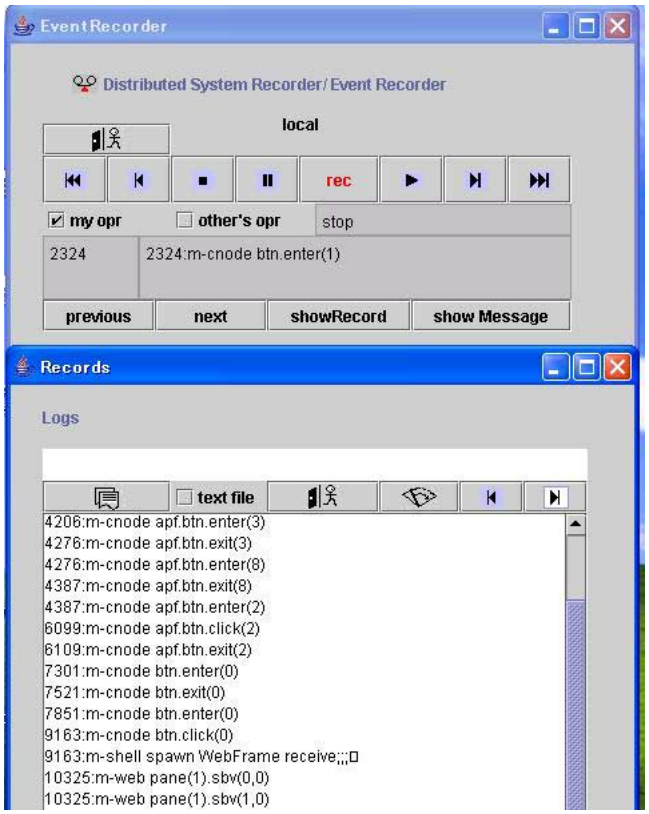

Figure 5. Event recorder/player and recorded commands

2.2.5 *State of Node Systems*

The teacher's node system has the special application which can control the operation state of student's node systems. The operation state includes the state of local operation (SLO) and the state of receiving commands (SCO). SLO is used for performing independent operations by node systems. SCO is used for performing the same operations by all node systems simultaneously.

2.3 Applications

DSR is equipped with applications by it self. The user of DSR doesn't need to purchase other applications for the benchmark test. The user can compare computer laboratories' equipment using the same applications. In order to realize the similar situation in a class, we are building simple but frequently used applications. Figure 6 shows the draw and the programming environment. A geometrical figure is drawn on the draw by the program on the programming environment. A picture is also drawn on the draw. Figure 7 shows the Web browser and the writer's assistant. This writer's assistant assists the user to write English sentences by showing frequencies of phrases in the inputted sentence graphically. A search engine is used for getting the frequencies.

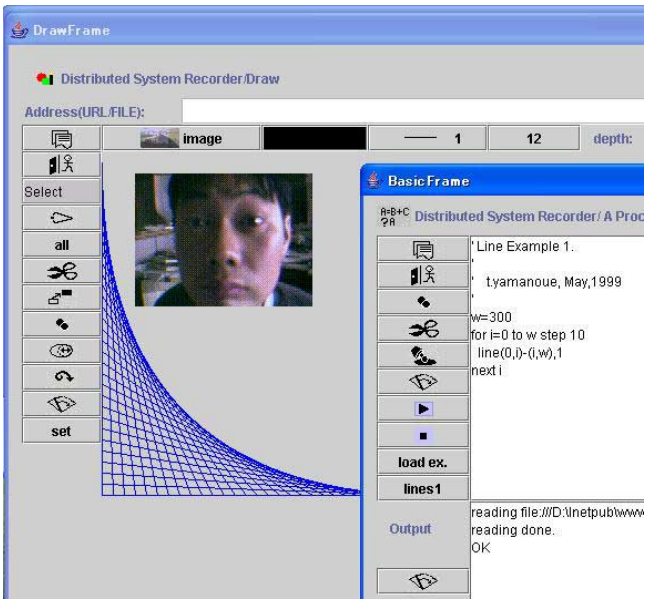

Figure 6. Draw and programming environment

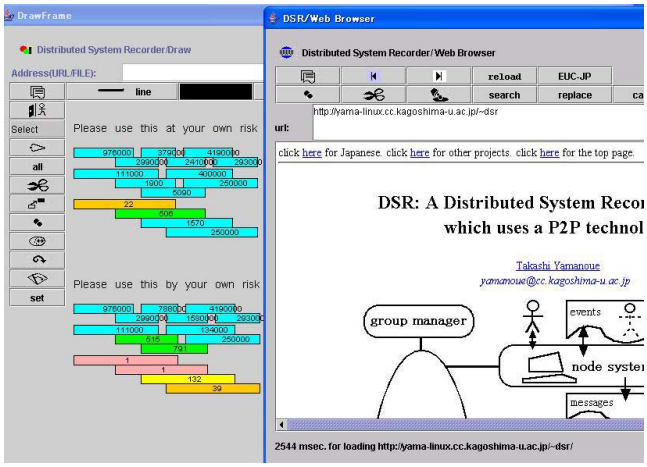

Figure 7. Web Browser and Writer's Assistant

3. EXPERIMENTS

DSR can let different kinds of distributed systems to do the same operation. DSR also can let the many terminal computers to do the same operation simultaneously. In order to verify these functions of DSR, We did the following experiment.

This experiment compares the performance of six kinds of distributed system by doing the same operations using DSR. This experiment is executed in the following steps.

1. Record operations on a computer using DSR. These operations are reading and writing data between client computers and a file server.
2. Play the recorded operations on 6 kinds of distributed system. Four of them have more than one client computer. In the case of these multiple client environment, the operations are executed simultaneously on these client computers. We couldn't find out any delay of the operation by the sight.

The recorded operations are the followings. These are executed in this order.

1. Read a 7KB GIF picture from the one file, on the draw of DSR.
2. Read a 16KB JPG picture from the one file, on the draw.
3. Read a 37KB JPG picture from the one file, on the draw.
4. Read a 3KB text which represent a figure from the one file, on the draw
5. Write the 3KB text to each user file, on the draw.
6. Read the 3KB text from the each user file, on the draw.
7. Read a 10KB text which represent a figure from the one file, on the draw
8. Write the 10KB text to each user file, on the draw.
9. Read the 10KB text from the each user file, on the draw

The six kinds of distributed system are the followings

PC-1: One laptop computer. This computer did the role of not only the client computer but also the file server.

PC-2: One desktop computer and one laptop computer. The desktop computer did the role of the file server and the role of a client computer. The laptop computer did the role of a client computer. Two computers were connected by a switch with 100Mbps full duplex connection.

Linux-1: One Linux thin client computer (one node) and a file server machine. A Linux thin client computer is a terminal computer which has no hard drive and its OS is Linux. They are connected with a switching network. The client is connected to a switch with 100Mbps full duplex connection. The file server machine is connected with 1Gbps full duplex connection.

Linux-2: Two Linux thin client computer (two nodes) and a file server machine. They are connected as same as the Linux-1 environment.

Linux-10: 10 Linux thin client computer (10 nodes) and a file server machine. They are connected as same as the Linux-1 environment.

Linux-75: 75 Linux thin client computer (75 nodes) and a file server machine. They are connected as same as the Linux-1 environment.

A shot of the experiment of Linux-75 is shown in Figure 8. The detail of each distributed system is shown in Figure 9.

The result of the experiment is shown on the Table 1. We could let the different distributed system to do the same operation whether the client computer's OS was Windows or Linux. PC-1's performance was worse than PC-2. We think that it is because the performance mainly depends on the performance of the hard drive or the file server.

Table 1. Performance of 6 kinds of distributed system by doing the same operations using DSR

			Time of the Performance of the Operation (sec.)					
No.	**Operation**		**PC-1**	**PC-2**	**Linux-1**	**Linux-2**	**Linux-10**	**Linux-75**
1.	Reading a 7KB GIF picture	Max.	4.1	6.2	0	1	5.3	5.9
	from the same file.	Min.	4.1	0.4	0	0.3	0	0
		Ave.	4.1	3.3	0	0.7	3.7	5.1
2.	Reading a 16KB JPG picture	Max.	11.6	10.2	11.8	11.4	9.1	9.5
	From the same file.	Min.	11.6	0.7	11.8	9.2	0.7	0.1
		Ave.	11.6	5.5	11.8	10.3	8.1	8
3.	Reading a 37KB JPG picture	Max.	14.6	1.1	0.1	16.1	21	20.5
	from the same file	Min.	14.6	1.2	0.1	0	0	0
		Ave.	14.6	1.2	0.1	8.1	14.3	13.5
4.	Reading a 3KB text	Max.	0.6	0.5	0.6	0.2	0.2	0.3
	from the same file	Min.	0.6	0.4	0.6	0.1	0	0
		Ave.	0.6	0.5	0.6	0.2	0.1	0.1
5.	Writing the 3KB text	Max.	0.9	0.9	0	0.2	0.3	0.4
	to each user file	Min.	0.9	0.4	0	0.1	0.1	0
		Ave.	0.9	0.7	0	0.2	0.2	0.2
6.	Reading the 3KB text	Max.	0.2	0.1	0.2	0	0.2	0.2
	from each user file.	Min.	0.2	0	0.2	0	0	0
		Ave.	0.2	0.1	0.2	0	0	0
7.	Reading a 10KB text	Max.	0.6	0.2	0	0	0.2	0.3
	from the same file.	Min.	0.6	0	0	0	0	0
		Ave.	0.6	0.1	0	0	0	0.1
8.	Writing the 10KB text	Max.	1.7	1.4	2.3	1.6	2.4	2
	to each user file	Min.	1.7	0.8	2.3	1.4	1.1	0.9
		Ave.	1.7	1.1	2.3	1.5	1.4	1.4
9.	Reading the 10KB text	Max.	0.2	0.2	0.2	0	0.2	0.3
	from each user file	Min.	0.2	0	0.2	0	0	0
		Ave.	0.2	0.1	0.2	0	0.1	0.1

Figure 8. Experiment with 75 clients

The performance of Linux-75 is not so worse than Linux-10. We think, it is because the cache memory of the file server machine reduces the access time to the hard drive. We also measured the latency of mouse moving. There was almost no delay in the mouse moving.

4. RELATED WORKS

LoadRunner[5] is a similar test system to DSR in the sense of that it measures the performance of a distributed system by emulating real-life user loads. LoadRunner can emulate hundreds or thousands of concurrent virtual users using minimal hardware resources. On the other hand, DSR has to have real hardware resources in order to measure the performance. However, DSR can test the real system by replaying real users' operations. When a distributed system is tested, operations of users are reproduced on the applications of DSR on the terminal computers of the distributed system. DSR can also measure the performance of client computers.

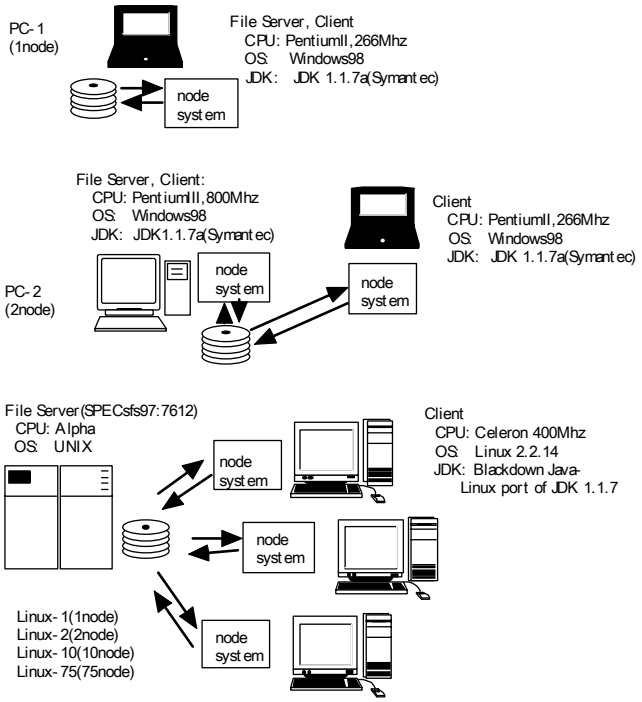

Figure 9. Six kinds of Distributed System which are used in this experiment

DBS (Distributed Benchmark System)[6] is also a similar test system to DSR in the sense of that it can measure the performance of a network from many points. DBS can measure the performance of entire TCP functions where DSR can not. However, DSR can measure the total system performance of a distributed system.

5. CONCLUDING REMARKS

We show DSR, a Distributed System Recorder, with its structure and an experiment. It records user operations on a distributed system, replays the operations on other distributed systems and shows the performance data. It also can let every computer doing the same thing simultaneously. DSR can be used on various kinds of distributed system because it is a platform independent.

One of the most drawbacks of DSR is that DSR can't measure the performance of a distributed system with using other applications which are not equipped with DSR. However, this drawback makes the DSR platform independent.

DSR doesn't have a spreadsheet and a mail reader/writer now. They should be equipped with DSR as its applications because they are popular applications in a school and daily life.

DSR generates a simple sequence of commands. In order to interact with Web CGI and other interactive applications on the network by the application of DSR, the sequence of the commands must be more intelligent.

There are many bugs in DSR. We are debugging and improving DSR. We should have much more experiments with DSR.

6. ACKNOWLEDGMENTS

We thank to staff and students of Information Science Center, Kyushu Institute of Technology and Computing and Communications Center, Kagoshima University for their suggestions and help. Part of this work is supported by the Grant-in-Aid of the Ministry of Education, Science and Culture of Japan(C)(2)(09680401).

7. REFERENCES

[1] Brown, D. G., Burg, J. J., Dominick, J. L. A Strategic Plan for Ubiquitous Laptop Computing. *Communications of the ACM*, Vol. 41, Issue 1, pp.25-35, Jan. 1998.

[2] Green, K. C. Campus Computing 2000: The National Survey of Information Technology in U.S. Higher Education. *EDUCAUSE Information Resources Library,* EDU0035, 2000.

[3] Hirahara, T., Yamanoue, T., Anzai, H., and Arita, I. Sending an Image to a Large Number of Nodes in Short time using TCP. In *Proceedings of the ICME2000, IEEE International Conference on Multimedia and Exp (ICME2000),* New York City, USA, July 30-Aug.2,pp.987-990,2000.

[4] Kossuth, J. M. If I Could Only Start from Scratch: A Case Study of Infrastructure Design at Olin College. *EDUCAUSE Information Resources Library,* NCP0113, 2001.

[5] LoadRunner, *http://www.mercury.com/us/products/performance-center/loadrunner/*

[6] Maruyama, Y. DBS: a powerful tool for TCP performance evaluations. In *Proceedings of SPIE, Performance and Control of Network Systems,* Vol. 3231, Nov. 1997.

[7] Yamanoue, T., Minami, T., Ruxton, I., Sakurai, W. Learning Usage of English KWICly with WebLEAP/DSR. In *Proceedings of the 2nd International Conference on Information Technology and Applications (ICITA-2004),* 14-6, Harbin, China, 8-11, Jan. 2004.