# High-Resolution Streaming Functionality in SAGE2 Screen Sharing

Kazuya Ishida[1], Daiki Asao[2], Arata Endo[1], Yoshiyuki Kido[3], Susumu Date[3], and Shinji Shimojo[3]

[1] Graduate School of Information Science and Technology, Osaka University, Japan,
{kazuya.ishida, endo.arata}@ais.cmc.osaka-u.ac.jp
[2] School of Engineering, Osaka University, Japan,
asao.daiki@ais.cmc.osaka-u.ac.jp
[3] Cybermedia Center, Osaka University, Japan,
{kido, date, shimojo}@cmc.osaka-u.ac.jp

**Abstract.** Visualization on a Tiled Display Wall (TDW) is an effective approach for sharing large quantities of scientific data among researchers in collaborative research. SAGE2 (Scalable Amplified Group Environment) is a popular middleware for organizing multiple monitors into a TDW. SAGE2 has a useful function, Screen Sharing, which allows the user to utilize existing applications on a TDW without redevelopment. However, the current Screen Sharing has a problem in that it displays the application window at the same resolution as the monitor devices connected to the user's PC. Such insufficient resolution degrades the visibility of scientific data visualized on a TDW. In this paper, we incorporate a frame-streaming method we designed into Screen Sharing to realize the functionality of high-resolution streaming. Our evaluation demonstrates that our method enables Screen Sharing to display the application window at an arbitrary resolution on a TDW. Our method is also effective in improving the frame rate of Screen Sharing. For example, 19.2 fps is achieved when displaying a 4K application on a TDW.

**Keywords:** Visualization, Tiled Display Wall, SAGE2, Screen Sharing

## 1 Introduction

The collaboration of researchers is essential for scientific discovery in modern science. To deal with difficult and complex tasks, the collective effort of many researchers has often been required. For example, in the KBDD (K supercomputer-based drug discovery) project [1], researchers from various companies and academic institutions have worked together to discover chemical compounds for novel drugs through molecular dynamics simulations performed by the K supercomputer.

In collaborative research, researchers involved in a project often have discussions based on the scientific data they acquire. As a consequence of the recent

performance enhancement of computers and measuring equipment, the opportunities for researchers to treat large quantities of scientific data have been increasing. To facilitate research discussions using such large data, the data should be intuitively visualized and arranged on a high-resolution screen so that it can be easily understood and compared with other data by the researchers. In the KBDD project, for example, the results of the molecular dynamics simulations are supposed to be visualized in the discussions on the druggability of chemical compounds. Since the researchers discuss druggability while comparing the targeted compound with many different types of other compounds, the multiple visualized simulation results need to be laid out on a high-resolution screen [2]. For this purpose, visualization on a TDW (Tiled Display Wall) [3] is an appropriate approach. A TDW is a scalable visualization system, which can provide a high-resolution virtual screen by combining multiple sets of computers and monitors. A large-scale TDW allows a large number of researchers to observe the visualized data simultaneously and to exchange ideas with each other on the spot.

To construct a large-scale TDW, various middleware have been developed, such as SAGE2 (Scalable Amplified Group Environment) [4], CGLX (Cross Platform Cluster Graphics Library) [5] and DisplayCluster [6]. In particular, SAGE2 has been accepted in various research fields because of its advantageous features for both co-located and remote collaborative work. In reality, 91 or more universities/institutes all over the world have adopted SAGE2 [7].

The major reason why SAGE2 is suitable for collaborative research is that SAGE2 provides Screen Sharing, which is an optional function to leverage existing visualization applications on a TDW. Most middleware-based TDW can display only the applications developed with the special APIs provided by the middleware. To use an existing application on such a TDW, the user is required to add the API calls in the source code and re-compile it. In contrast, a SAGE2-based TDW can perform the mirroring of an application window generated on the user's PC by using Screen Sharing. This function allows the user to leverage a wide range of existing applications in a non-invasive manner.

However, the current Screen Sharing has an undesirable problem in that it cannot display an application window at an arbitrary resolution on a TDW. This is because Screen Sharing is realized by capturing video frames on the user's PC and streaming them to the TDW. The resolution of these video frames is determined by the size of a framebuffer on the user's PC. A framebuffer is a memory space to store image data drawn by applications. The available sizes for a framebuffer are confined to the specific values supported by the monitor devices connected to the user's PC video card device. Therefore, if the resolution which the connected monitor devices support is much lower than the TDW, the application window is displayed on the TDW with its visibility degraded.

In this paper, we propose a frame-streaming method which enables Screen Sharing to display a high-resolution application window regardless of the limitations of the connected monitor devices. In addition, our method improves the efficiency of the processing in streaming video frames to achieve a better

frame rate. The remainder of this paper is structured as follows. In section 2, we present the overview of SAGE2 and Screen Sharing. In section 3, we describe the technical problems and issues we tackled. In section 4, works related to our research are introduced. Section 5 shows the frame-streaming method for realizing high-resolution streaming functionality in Screen Sharing. The experiments for evaluating the validity and scalability of the proposed method are described in Section 6. In section 7, we conclude this paper and note our future work.

## 2 SAGE2 Screen Sharing

### 2.1 Architecture of SAGE2

SAGE2 (Scalable Amplified Group Environment) [4] is an open-source middleware, which has been developed at EVL (the Electronic Visualization Laboratory) at the University of Illinois, Chicago (UIC). SAGE2 can build a large virtual desktop on a TDW and arrange multiple application windows on it just like a window manager. In addition, SAGE2 has a function to distribute the visualized contents to other TDWs in geographically separated sites via a WAN (Wide Area Network).

SAGE2 is cloud and web-browser-based technology, which is implemented with HTML5, JavaScript and WebGL. Figure 1 shows the architecture of a general SAGE2-based TDW. A SAGE2-based TDW consists of three components: the *display clients*, the *interaction client* and the *SAGE2 server*. The display clients are the HTML pages opened by the full screen web browsers on all the monitors of the TDW. Each display client renders the particular domain of the virtual desktop corresponding to its own client ID. The interaction client is the web page with the JavaScript programs to provide SAGE UI, which is a graphical user interface of SAGE2. Through the SAGE UI, the user can move and resize the windows on the virtual desktop. The SAGE2 server is a web server that plays the core role of controlling the entire TDW. The SAGE2 server reflects the results of user operations on SAGE UI and synchronizes the other components by exchanging the control messages through a WebSocket protocol. The control message is composed of a message name and several data (e.g. window size and image data). When each component receives a control message, a callback process corresponding to its message name is executed on the component.

### 2.2 Screen Sharing

Screen Sharing is the function of SAGE2 for sharing the user's desktop contents among the members of the project. This function is realized by streaming video frames of the application window to the TDW with several browser APIs such as *getUserMedia()*, *drawImage()* and *toDataURL()*. Thanks to Screen Sharing, SAGE2 allows the user to leverage existing applications on a TDW without modification.

Screen Sharing is the function of SAGE2 for sharing the user's desktop contents among the members of the project. This function is realized by streaming

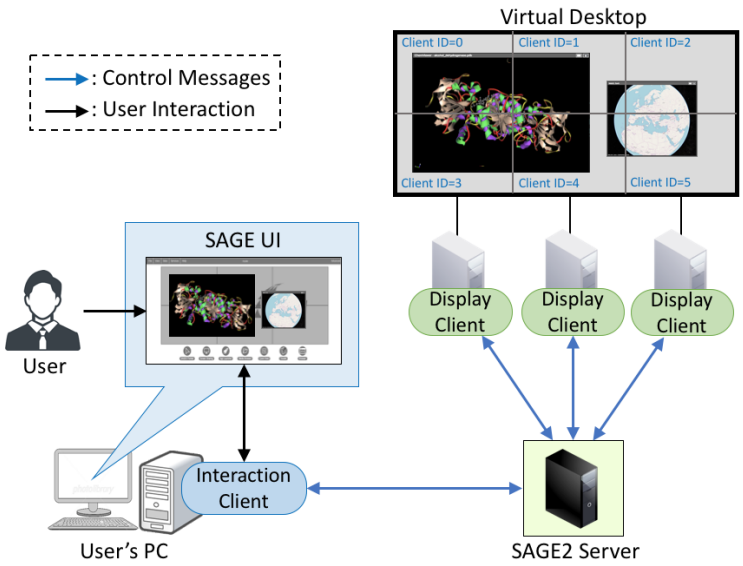



Fig. 1: Architecture of the SAGE2-based TDW

video frames of the application window to the TDW with the several browser APIs such as *getUserMedia()*, *drawImage()* and *toDataURL()*. Thanks to Screen Sharing, SAGE2 allows the user to leverage existing applications on a TDW without modification of them.

Figure 2 illustrates how Screen Sharing works. If the user launches Screen Sharing through SAGE UI, the WebSocket connection is established between the user's PC and the SAGE2 server by exchanging the *requestToStartMediaStream* message and the *allowAction* message. Next, the interaction client starts capturing video of the application window in the background process by using a *getUserMedia()* API. This API captures video from a framebuffer, which is memory space allocated on the graphics memory so that an X server can store image data drawn by X applications. The captured video is streamed to the hidden video element of the interaction client. Then, the loop for streaming the frames of the captured video is iterated continually until Screen Sharing is terminated by the user. This loop is composed of the following four steps ((1)∼(4) in Fig. 2).

(1) *Extraction*: The interaction client extracts the latest frame from the captured video, and places this frame onto the hidden canvas element by using the *drawImage()* API.
(2) *Conversion*: The extracted frame is converted to a JPEG image in Base64 format with the *toDataURL()* API.
(3) *Transmission*: The Base64 string of the frame is transmitted as the *updateMediaStreamFrame* message to the SAGE2 server.

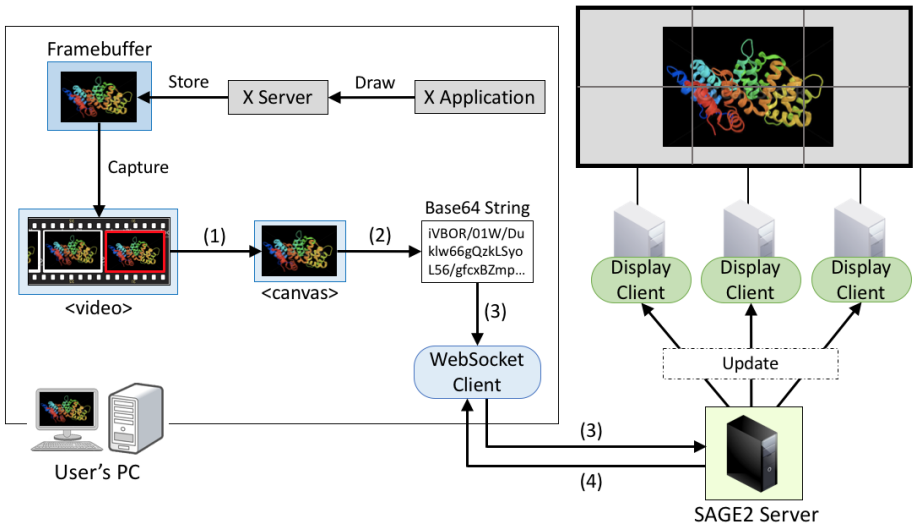



Fig. 2: System architecture for Screen Sharing

(4) *Synchronization*: After the SAGE2 server updates the frame to all the display clients, the SAGE2 server passes the *requestNextFrame* message to the user's PC.

## 3 Problems and Issues

The current Screen Sharing of SAGE2 has the following two problems: *Resolution Constraint* and *Conversion Delay*. In this section, we describe these problems and the technical issues required to overcome them.

### 3.1 Resolution Constraint

As mentioned in 2.2, the current Screen Sharing captures video frames of the application window from the framebuffer on the user's PC. The resolution of these video frames is determined by the size of the framebuffer on the user's PC. Generally, since the size of a framebuffer is fixed to the specific values which are supported by the monitor devices connected to the user's PC video card device, there is a limit to the resolution at which Screen Sharing can display the application on a TDW.

This *Resolution Constraint* problem causes the visibility of the application on a TDW to deteriorate when there is a large difference in the number of pixels between the user's PC and the TDW. An example of such a case is depicted in Fig. 3. In this example, the screen resolution of the user's PC is HD (1280x720) and the total resolution of the TDW is 7680x3240. In this case, Screen Sharing

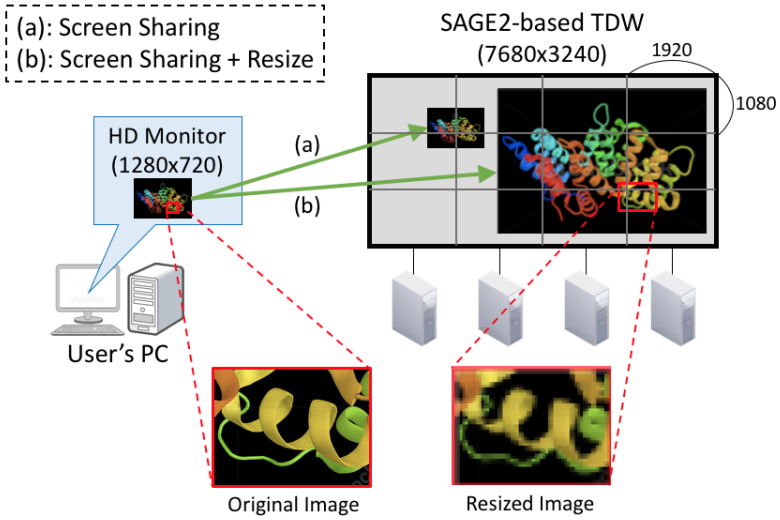



Fig. 3: Example of the deterioration of visibility caused by the *Resolution Constraint* problem

displays the application as a relatively small sized window on the TDW ((a) in Fig. 3). Although the user can resize this window to make it larger through SAGE UI, the enlarged window will become a rough image ((b) in Fig. 3). Such poor visibility becomes a hindrance to gaining information and insights from visualized scientific data. To improve the visibility of the application using Screen Sharing, a technical issue arises: namely, how a flexible framebuffer can be prepared on the user's PC without being constrained by the connected monitor devices.

### 3.2 Conversion Delay

In the current Screen Sharing, video frames captured on the user's PC are streamed to a TDW by iterating four steps: (1) *Extraction*, (2) *Conversion*, (3) *Transmission* and (4) *Synchronization*. In advance of this research, we measured the processing time to refresh one frame in the current Screen Sharing by varying its resolution. This precursor experiment was conducted on the environment detailed in 6.1.

The result of the precursor experiment is shown in Fig. 4. This graph suggests that each step requires more processing time as the resolution of video frames become higher. In particular, the processing time for step (2) (*Conversion*) sharply rises related to the increase of the resolution. In other words, the processing for JPEG compression and Base64 encoding causes a serious delay in Screen Sharing. This *Conversion Delay* problem leads to a significant degradation of the frame rate when Screen Sharing displays a high-resolution application window

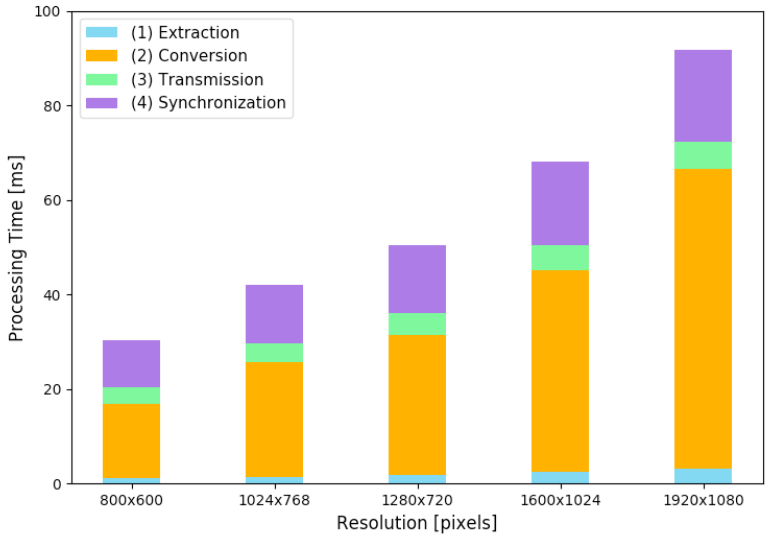



Fig. 4: Breakdown of the processing time to refresh one frame in the current Screen Sharing

on a TDW. To suppress the degradation of the frame rate in Screen Sharing, another technical issue arises: namely, how the delay time by the step (2) can be reduced.

## 4 Related Works

We have focused on displaying an existing application on a middleware-based TDW without source code modification. Some other researchers have also worked on this topic.

Tada *et al.* proposed an adapter solution for SAGE [8]. SAGE (Scalable Adaptive Graphics Environment) [9] is a middleware that is a predecessor of SAGE2. Unlike SAGE2, SAGE does not provide Screen Sharing. To utilize an existing application on a SAGE-based TDW, developers have to add many APIs provided by SAGE like the other middleware. Their adaptor solution allows the users to display a window of the existing application on a TDW without such troublesome work. As with our method, their solution adopts a virtual framebuffer to capture image data drawn by applications. However, our research also pursues a method to improve the frame rate in a streaming high-resolution window.

Kimball *et al.* developed a framework to stream a user's desktop contents to a CGLX-based TDW [10]. CGLX (Cross Platform Cluster Graphics Library) [5] is a middleware for building a TDW, which has an OpenGL-based distributed rendering architecture. In their framework, captured frames are compressed with

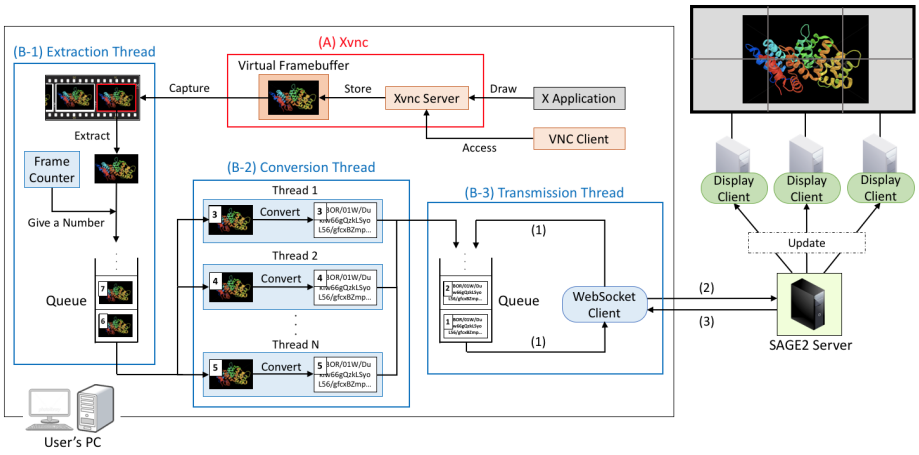



Fig. 5: System architecture for the Screen Sharing with the proposed method

H.264 video encoding and delivered to a TDW using UDP multicast. Their research differs from our research in that they aim to achieve low bandwidth and low latency in the desktop streaming for CGLX. Our research aims to incorporate a high-resolution streaming functionality into the Screen Sharing of SAGE2.

Neal *et al.* implemented ClusterGL, which is a system to make existing OpenGL applications available on a TDW [11]. ClusterGL captures OpenGL commands from OpenGL applications and distributes them with their arguments to the renderers on display nodes. To enhance its performance, ClusterGL also performs several optimizations such as frame differencing and data compression. ClusterGL has a disadvantage in that it can be used only for limited applications because it supports only OpenGL 2.1 or earlier. In contrast, SAGE2 Screen Sharing with our method can be applied to a wider range of applications because it adopts not command streaming but frame streaming.

## 5 Proposed Frame-Streaming Method

### 5.1 Overview of Proposed Method

We proposed the frame-streaming method for achieving the issues described in section 3. Figure 5 overviews the system architecture for Screen Sharing with the frame-streaming method we designed. The red and blue squares in Fig. 5 are the implementation to incorporate the proposed method in Screen Sharing.

First, a Xvnc server is launched on the user's PC ((A) in Fig. 5). The user has to direct the X application to draw image data on the virtual framebuffer of the Xvnc server, then access it via the VNC client. Next, the WebSocket connection is established between the user's PC and the SAGE2 server in the same way as the current Screen Sharing. After that, the procedures for streaming video frames

are repeated asynchronously by the three types of threads: the *extraction thread*, the *conversion threads* and the *transmission thread* ((B-1)∼(B-3) in Fig. 5). The extraction thread is in charge of capturing the video from the virtual framebuffer and extracting the latest frame from it. The extracted frames are given serial numbers and stored in its queue. The conversion threads undertake the parallel frame conversion. Each conversion thread gets one frame from the queue in the extraction thread and converts it to a JPEG image in Base64 format. The converted frames are collected in the queue of the transmission thread. The transmission thread repeatedly sends the converted frames to the SAGE2 server according to the following three procedures ((1)∼(3) in Fig. 5).

(1) The converted frame is retrieved from the queue and its serial number is checked. If the number is not the one which the next frame should have, the frame is returned to the queue and the procedure (1) is redone.
(2) The Base64 string of the retrieved frame is transmitted as the *updateMediaStreamFrame* message to the SAGE2 server.
(3) The thread waits until the *requestNextFrame* message is sent back from the SAGE2 server.

### 5.2 Xvnc

To achieve a solution to the *Resolution Constraint* problem, we applied Xvnc [12]. Xvnc is a VNC (Virtual Network Computing) server that can also act as an X server. The comparison between an X server and an Xvnc is shown in Figure 6. The Xvnc is different from the usual X server in that the Xvnc uses a virtual framebuffer instead of a normal framebuffer. A virtual framebuffer is alternative memory space allocated on the shared memory for off-screen rendering. Xvnc stores image data drawn by X applications on its own virtual framebuffer, and does not output the frames to the connected monitor devices. To see and access X applications displayed on the virtual framebuffer, the user has to access them via a VNC client.

As opposed to a normal framebuffer, the size of virtual framebuffer can be optionally changed independent of the specification of the monitor devices. Therefore, Xvnc allows the user's PC to prepare the application window drawn at a larger resolution than the connected monitor devices can handle. The available resolution in the current Xvnc ranges from 32x32 to 32768x32768, which is the sufficient support for displaying an application window on a TDW.

### 5.3 Pipeline Streaming

To solve the issue for the *Conversion Delay* problem, we designed *pipeline streaming*: a concept for reducing wait time that occurs due to the processing for *Conversion*. Figure 7 illustrates how pipeline streaming works. The current Screen Sharing performs *sequential streaming*: the four steps for streaming video frames are iterated sequentially as shown in (a) in Fig. 7. In sequential streaming, refreshment of the TDW slows down greatly with respect to the increase of the resolution because of the processing for *Conversion*. This delay can be prevented by

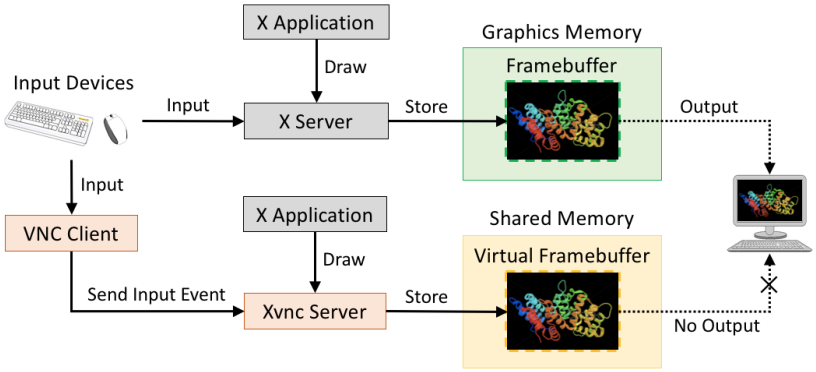



Fig. 6: Comparison between X server and Xvnc

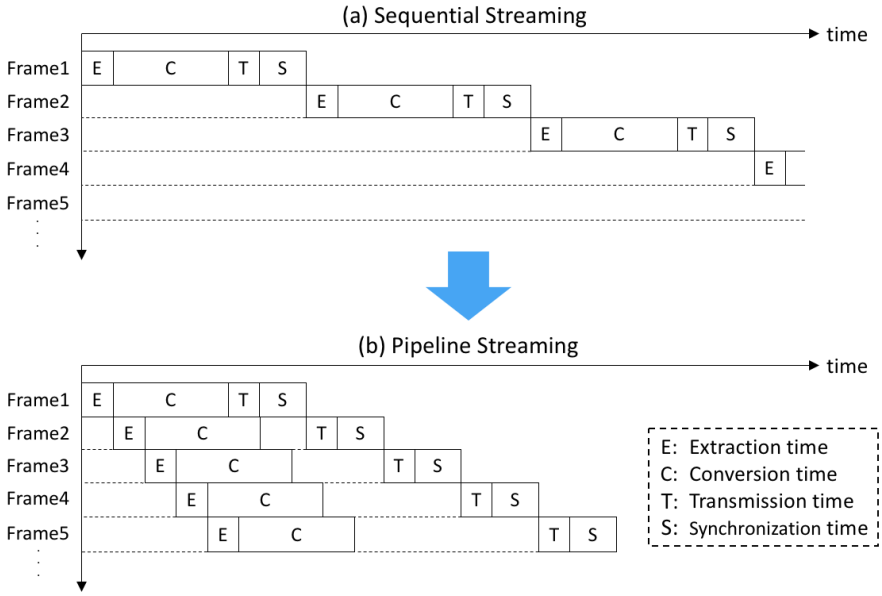



Fig. 7: Concept of pipeline streaming

pipelining streaming in Screen Sharing as in (b) in Fig. 7. In pipeline streaming, *Extraction* and *Conversion* for the succeeding frames are executed antecedently during the process for the current frame. Moreover, each processing for *Conversion* is executed simultaneously with the other ones by using thread-level parallelism for further improvement of the efficiency.

Table 1: Node specifications of the TDW

| Name | Specification |
|---|---|
| CPU | Intel Xeon E5-2640 (2.5GHz) ×2 |
| GPU | NVIDIA Quadro K5000 ×1 |
| Memory | 64GB |
| OS | CentOS 7.2 |

Table 2: Software versions used in the evaluation

| Name | Version |
|---|---|
| SAGE2 | 3.0.0 |
| Chromium Browser | 65.0.3325.181 |
| TurboVNC Server | 2.1.2 |
| TurboVNC Viewer | 2.1.2 |
| Cytoscape | 3.6.1 |

## 6 Performance Evaluation

We conducted two experiments to evaluate the validity and the scalability of Screen Sharing with the proposed method.

### 6.1 Evaluation Environment

For the evaluation, we used the 24-screen Flat Stereo Visualization System in the Cybermedia Center, Osaka University [13]. This TDW is composed of seven nodes (one head node and six display nodes), each of which has the specification as shown in Table 1. All nodes composing the TDW are connected to a dedicated 10 Gigabit Ethernet switch. The head node has a Full HD (1920x1080) monitor and input devices (a mouse and a keyboard) and each display node is connected to four Full HD monitors arranged in tandem.

Figure 8 shows the evaluation environment used in the experiments. One display node was used as both the SAGE2 server and the display client. The other display nodes were used only for the display clients. All the display clients were accessed via Chromium browsers [14]. The head node became a client node to launch Screen Sharing with the proposed method. To use Xvnc, a TurboVNC server and viewer [15] were installed on the client node. The JPEG compression quality of the video frames was configured as 90. We used Cytoscape [16] as the application displayed on a TDW. Cytoscape is a representative software to visualize complex networks such as molecular interaction networks and biological pathways. The versions of these software are presented in Table 2.

### 6.2 Observation of Visibility

To confirm the improvement of the visibility of the application, we displayed the window with Cytoscape at the maximum resolution of the TDW (11520x4320)

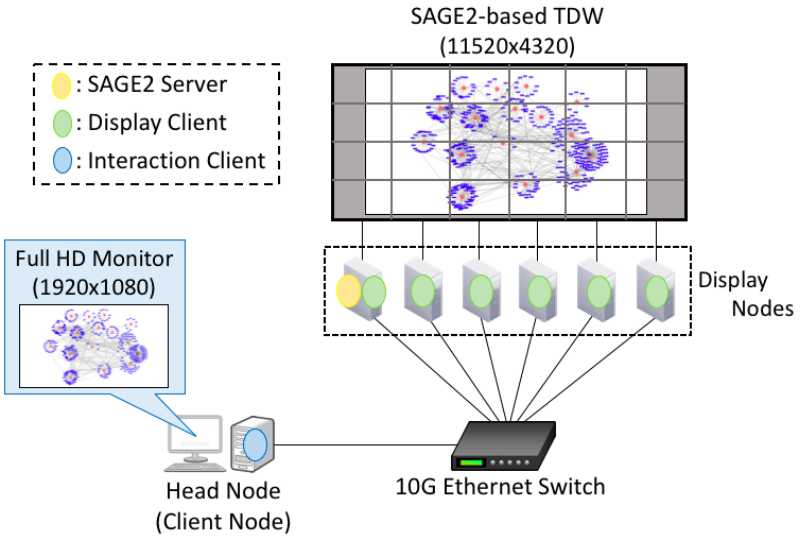



Fig. 8: Evaluation environment

using the Screen Sharing with the proposed method. Fig. 9 is a snapshot of the Cytoscape on the TDW. We also displayed the same window using the current Screen Sharing and the resize operation, and compared them.

From (a) and (b) in Fig. 10, it can be seen that the proposed method enables Screen Sharing to display the application window more precisely. The key difference between (a) and (b) is the visibility of the label strings. In (b), the label strings became blurry as a result of the resize operation. By contrast, all the label strings in (a) are clearly displayed. This result shows that the proposed method is effective in improving the visibility of the application window displayed by Screen Sharing.

### 6.3 Frame Rate Measurement

To investigate the scalability of Screen Sharing with the proposed method, we measured the frame rate by varying the resolution of the window with Cytoscape. The number of the conversion threads was also changed to 1∼4. In addition, the frame rate achieved by Screen Sharing with the sequential streaming (i.e. the current Screen Sharing) was also surveyed.

The result of the measurement is depicted in Fig. 11. This graph indicates that the Screen Sharing with the proposed method outperforms the Screen Sharing with the sequential streaming at all the resolution which we measured. Moreover, it is implied that the frame rate is improved by increasing the number of the conversion threads. The proposed method enabled Screen Sharing to achieve 19.2 fps (frames per second) when displaying the window at 4K resolution (3840x2160).

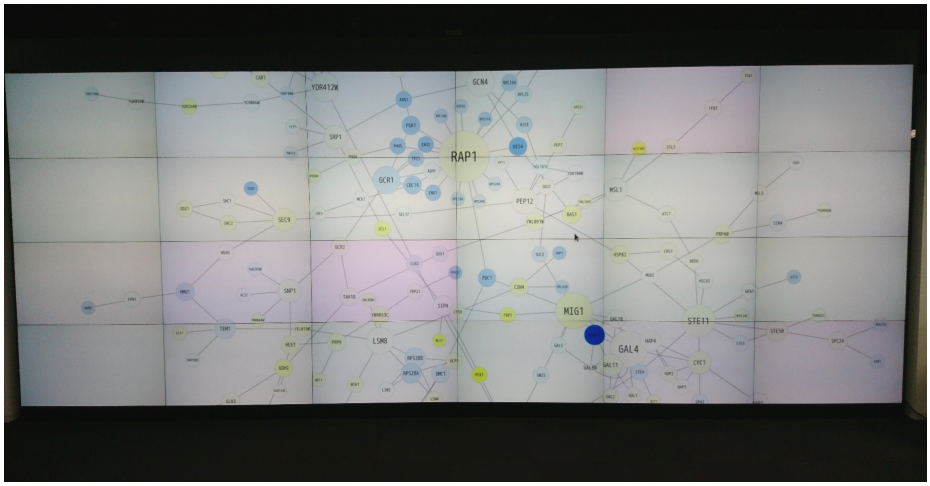


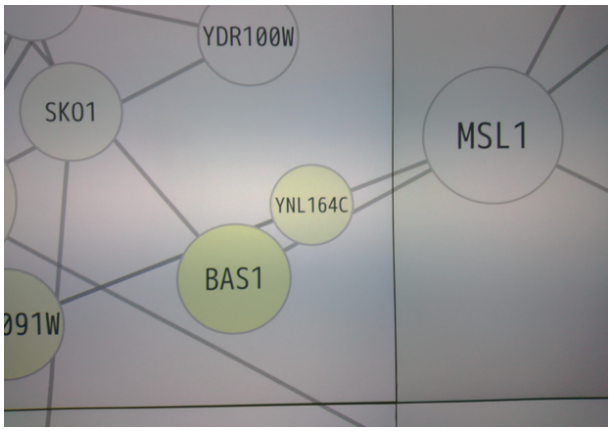

Fig. 9: Cytoscape on the TDW

### 6.4 Discussion

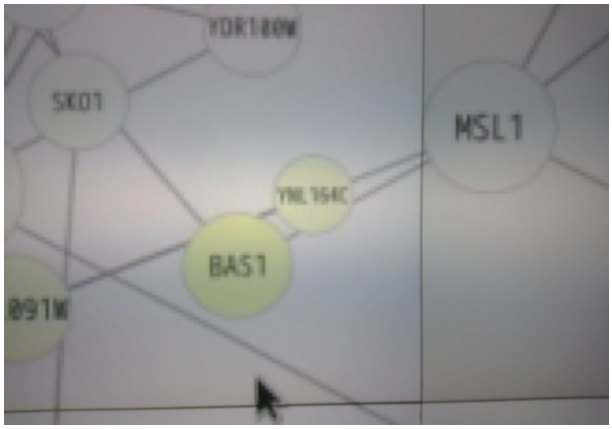
6.2 and 6.3 demonstrated that our proposed method enabled Screen Sharing to stream a high-resolution application window at a better frame rate. However, there is still room for improvement in the proposed method for more practicality. The points to be improved are discussed below.

First, it is necessary to make the proposed method available on the platforms other than X Window System. The current proposed method can be applied only to X applications because of Xvnc. To display a wider range of applications on a TDW, optional kinds of VNC servers have to be leveraged in the proposed method. For example, the Xvnc server should be replaced with a TightVNC server [17] to display Windows applications.

Second, further extension of pipeline streaming is needed to enhance the frame rate when streaming a high-resolution window. In Fig. 11, the degree of improvement of the frame rate becomes smaller as the resolution increases. There are two reasons for this result. Firstly, the processing time for *Conversion* becomes too large to be suppressed by pipelining. To realize maximum efficiency, *Conversion* for each frame should be completed by the end of *Synchronization* for the previous frame. Nevertheless, it becomes impossible to achieve this requirement when streaming the high-resolution window because of the rapid increase of the processing time for *Conversion*. To deal with this inconvenience, accelerating *Conversion* in itself is necessary. For example, it is effective to re-implement the processing for *Conversion* by using the primitive programming languages such as C/C++ or Fortran. Secondly, the processing time for *Transmission* and *Synchronization* becomes so large that it cannot be ignored. When the resolution of the video frames is high, a serious delay is likely to be caused by the network congestion in *Transmission* and *Synchronization*. This situation slows




(a) With the proposed method




(b) Without the proposed method

Fig. 10: Comparison of visibility on the TDW

down the entire streaming because *Transmission* and *Synchronization* cannot be pipelined. To cope with this situation, it is essential to reduce the network traffic of the streaming with some techniques such as data compression.

## 7 Conclusion

In this paper, we have developed a frame-streaming method to realize the high-resolution streaming functionality in Screen Sharing. The evaluation in this paper demonstrated that our proposed method enables Screen Sharing to stream a high-resolution application window regardless of the specifications of the connected monitor devices. The evaluation also showed that our proposed method improves the frame rate with pipeline streaming.

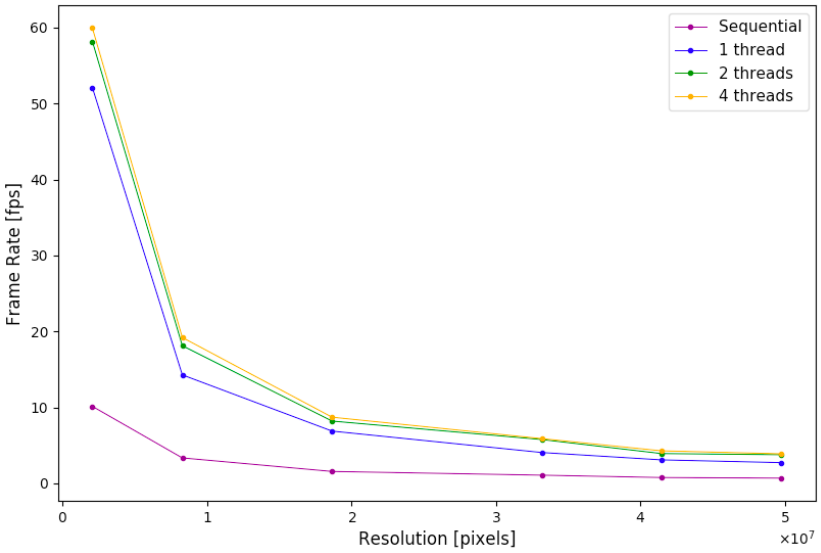



Fig. 11: Frame rate of Screen Sharing

For our future work, we will tackle the new problems associated with the proposed method described in 6.4. Through this effort, we aim to make SAGE2 more applicable to various research areas.

## Acknowledgment

This work was supported by the JSPS KAKENHI Grant Number JP18K11355 and the "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" in Japan (Project ID: jh160056-ISH, jh170056-ISJ, jh180077-ISJ).

## References

1. Brown, J.B., Nakatsui, M., Okuno, Y.: Constructing a Foundational Platform Driven by Japan's K Supercomputer for Next-Generation Drug Design. Molecular Informatics **33**, 732–741 (2014)
2. Lau, C.D., Levesque, M.J., Chien, S., Date, S., Haga, J.H.: ViewDock TDW: High-throughput visualization of virtual screening results. Bioinfomatics **26**(15), 1915–1917 (2010)
3. Humphreys, G., Buck, I., Eldridge, M., Hanrahan, P.: Distributed rendering for scalable displays. In: Proceedings of the 2000 ACM/IEEE Conference on Super-Computing, 30 (2000)
4. Marrinan, T., Aurisano, J., Nishimoto, A., Bharadwaj, K., Mateevitsi, V., Renambot, L., Long, L., Johnson, A., Leigh, J.: SAGE2: A new approach for data intensive collaboration using scalable resolution shared displays. In: Proceedings of

the 2014 IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, pp. 177–186 (2014)
5. Doerr, K.U., Kuester, F.: CGLX: A scalable, high-performance visualization framework for network displays. IEEE Transactions on Visualization and Computer Graphics **17**(3), 320–332 (2011)
6. Johnson, G.P., Abram, G.D., Westing, B., Navr'til, P., Gaither, K.: DisplayCluster: An interactive visualization environment for tiled displays. In: Proceedings of the 2012 IEEE International Conference on Cluster Computing, pp. 239–247 (2012)
7. Community - SAGE2. http://sage2.sagecommons.org/community-2
8. Tada, T., Date, S., Shimojo, S., Ichikawa, K., Abe, H.: A visualization adapter for SAGE-enabled tiled display wall. In: Proceedings of the 2011 IEEE International Conference on Granular Computing, pp. 613–618 (2011)
9. Renambot, L., Jeong, B., Jagodic, R., Johnson, A., Leigh, J.: Collaborative visualization using high-resolution tiled displays. In: Proceedings of the 2006 ACM CHI Workshop on Information Visualization Interaction Techniques for Collaboration Across Multiple Displays, pp. 1–4 (2006)
10. Kimball, J., Wypych, T., Kuester, F.: Low bandwidth desktop and video streaming for collaborative tiled display environments. Future Generation Computer Systems **54**, 336–343 (2016)
11. Neal, B., Hunkin, P., McGregor, A.: Distributed OpenGL rendering in network bandwidth constrained environments. In: Proceedings of the 2011 Eurographics Symposium on Parallel Graphics and Visualization, pp. 21–29 (2011)
12. Xvnc - the X-based VNC server. https://www.hep.phy.cam.ac.uk/vnc_docs/xvnc.html
13. Cybermedia Center, Osaka University large-scale visualization system. http://vis.cmc.osaka-u.ac.jp
14. Reis, C., Gribble, S.D.: Isolating web programs in modern browser architectures. In: Proceedings of the 2009 ACM European Conference on Computer Systems, pp. 219–232 (2009)
15. TurboVNC — Main / turboVNC. https://www.turbovnc.org
16. Shannon, P., Markiel, A., Ozier, O., Baliga, N.S., Wang, J.T., Ramage, D., Amin, N., Schwikowski, B., Ideker, T.: Cytoscape: A software environment for integrated models of biomolecular interaction networks. Genome Research **13**(11), 2498–2504 (2003)
17. TightVNC: VNC-compatible free remote control / remote desktop software. https://www.tightvnc.com