A Real-Time Collaborative Mechanism for Editing a Web Page and its Applications

Ryota Inoue, Yudai Kato, Takushi Goda
Department of Computer Science
Nagoya Institute of Technology
Nagoya, Japan
e-mail: {inouer, kyu, godata}@toralab.org

Tadachika Ozono, Shun Shiramatsu, Toramatsu Shintani
Department of Computer Science and Engineering
Graduate School of Nagoya Institute of Technology
Nagoya, Japan
e-mail: {ozono, sira, tora}@toralab.org

Abstract—We showed WFE, a real-time collaborative editing (RTCE) mechanism, in which a group of people can edit a web page on a web browser while members of the group share editing contents in real-time. In addition, we proposed applying WFE to a cloud computing environment, and we call the applied application WFE-S. The aim is to improve the system to have high scalability, elasticity, high responsiveness, and free it from maintenance. In this paper, we indicate further detailed implementation of WFE-S and Web applications using WFE-S. We showed that WFE-S enables us to make it easy to develop collaborative work systems as Web applications.

Keywords—cloud computing; collaborative workspace; web page editing; differential update mechanism;

I. INTRODUCTION

Collaborative work systems allow a physically dispersed group of people to engage in common task by providing an interface to a shared workspace. Over the past few years several studies have been made on collaborative work systems using web, and there has been renewal of interest in this field based on the web technology [1,2]. Such collaborative work system is one of the important applications of cloud computing. We proposed WFE, a system for Real-Time Collaborative Editing (RTCE) for existing Web page. The studies aim to enable several users to synchronously edit existing Web pages on each browser, while they share editing contents with each other in real-time. Figure 1 shows an example of performing synchronous editing using WFE from the user’s point of view. In WFE, users can edit a Web page with text editing, insertion of comments, image upload, table modification, and so forth. The editing features are accessed from the control panel shown by right click. A user modifies the contents on the right Web browser in Figure 1, and the modification is applied to the left Web browser immediately.

We proposed applying WFE to a cloud computing environment and its advantages [3,4]. The environment chosen is Google App Engine (i.e., GAE), a foundation for developing web applications provided by Google. GAE is a foundation for developing web application on which the applications use Google infrastructure easily without the hassle of hardware management. We will call the re-implemented application WFE-S in the rest of this paper. We have been trying to enable users to edit any Web pages that have no dynamic contents. We indicate further detailed implementation of WFE-S and its applications.

The paper is organized as follows. Section II describes our current system’s architecture and the solution of limitations in the preceding research. In Section III, we show the architecture of push system in detail. Section IV shows how to create collaborative work system based on WFE-S. We show that WFE-S enables us to make it easy to develop collaborative work systems as Web applications in Section V, and the system evaluation is performed in Section VI. Finally, Section VII presents our conclusions.

II. WFE-S: REAL-TIME COLLABORATIVE EDITING SYSTEM

We introduce the architecture of our current system and that the system solves the problems of previous WFE research. WFE is a system for Real-Time Collaborative Editing of existing Web pages based on Web agent technology [5]. Web agents on different Web browsers can synchronize Web contents by collaborating with agents on server.
A. System Architecture

An outline of our current system’s architecture is shown on Figure 2. The system is composed of a page identification and creation module, a page HTML fetch from a data store module, a page diff fetch module, a push reflection module, and a Web agent.

Our system can be used by adding a bookmarklet to a browser’s bookmarks. When the user clicks the bookmarklet, a Web agent is loaded from the server and the agent sends the current Web page’s URL to the page identification and creation module to translate the current Web page to RTCE Web page. The module confirms whether a Web page for RTCE is exists in the data store, and if the corresponding page does not exists, it is created. When a creation occurs, the identification and creation module fetches the original HTML of a Web page to be edited using a received URL, and translates it to editable HTML. The modifiable HTML is given a unique ID as the filename and then stored to the data store. We describe the detail of HTML creation in Section IV. Then, a URL of the page is returned as the form that says http://wfessynchroshare.appspot.com/{page identifier}. {page identifier} is the page’s filename, an unique ID that the page identification and creation module assigns when the page is generated.

The Web agent sends a HTTP request to the page HTML fetch from a data store module with the URL of a RTCE page. The module retrieves the page HTML from the data store and returns it to the client. This results in seamless transitions from the original page to its RTCE version by only one click on the bookmarklet. The Web agent also makes a request to the page diff fetch module. The module fetches HTML changes of the RTCE page from the data store and returns them to the client. The changes are applied to the RTCE HTML thereby editing history is reflected.

In a RTCE page, the users can use editing features, which is allowed by a preceding research’s system (text editing, insert of comments, image upload, table modification, and so on). When a modification occurs, a changed part of HTML and its selector, an HTML element identifier that is expressed in the form defined by Selectors API, to specify the parts that are sent to the push reflection module on the GAE server. These changed part and identifier are stored to the data store, and the module pushes them to the clients. Using the information of the edit, each Web agent on clients overwrites the HTML, thus enabling page synchronization.

B. Real-Time Performance

We need to improve WFE’s real-time performance because WFE has synchronization gap problem, which is used polling from the client side to detect changes in an HTML file on the server and synchronize the state of Web page in the client’s browsers. In the model of polling on WFE, each client performs getting notifications of an HTML file at each timing. There was, thus, a different time offset for each user until overall synchronization. We refer to it as synchronization gap. Because of synchronization gap, editing results can be lost, if editing is settled at approximately same time. Although
lessening the polling interval time can solve this problem, the approach leads to other negative effects in scalability.

In order to improve WFE’s real-time performance, WEF-S uses a push mechanism based on Channel API, GAE’s push facilities, for page synchronization. When editing, the changes are sent to GAE server and returned using channels built with Channel API. In our system, the channels are grouped into each RTCE page to distinguish destinations for changes. We detail the implementation in Section III. While each client has its timing for confirmation of update in polling, synchronization using push mechanism is immediate. Thus, the synchronization gap can be reduced.

Over and the above, to close the synchronization gap, we reduce amount of data used for synchronization. To achieve the data reduction, our system performs the synchronization of an edited part by using only information of modified element. This is described in Section III in detail.

C. Accessibility Hindrance

The accessibility hindrance of WFE is server construction with a WFE server application for modified Web pages. This impacted potential users, namely companies without an IT department or employees qualified in server administration. Such companies usually rely on mutualized hosting for their Websites. Thus, in order for employees to perform RTCE on the design and contents of the company’s Web pages using WFE, the company would need to at least outsource server administration to another company, which can be troublesome.

To solve WFE’s accessibility hindrance, WFE-S uses a bookmarklet that simplifies an operation to begin RTCE. Using this bookmarklet on a Web page, the page is taken into GAE server and the corresponding RTCE page is generated if it does not exist. This operation also performs page transition to the RTCE page automatically. In other words, due to seamless copy of an original Web page to the GAE server and the forwarding process, the users are released from WFE’s accessibility issue.

III. SYNCHRONIZATION MECHANISM BASED ON PUSH DELIVERY METHOD

We explain the implementation of our differential push mechanism for RTCE. A push mechanism in our system is based on the Channel API. This section details how it functions. Figure 3 shows a sequence diagram representing our push system. The push mechanism is comprised of two stages, a channel creation stage and a modification notice stage. User A and User B are RTCE users on a same page, and User B already built the channel. A Web agent performs a client side process of push mechanism.

A. Push System Flow

By accessing the RTCE page, the client creates a communication channel with the server for push operation. Because a token generated by the server is needed for the creation of a channel, an AJAX request is sent to a token issuance module on GAE server. The token must be grouped to settle push destinations, thereby the file name of the RTCE page is sent to the server as User A’s Web agent in the Figure 3. The token issuance module issues the token related to the file name, specifically a create_channel method is called with the file name, and returns the token to the client. When receiving, the client creates the channel and opens a socket for changes reflection.

Push operations from the server are performed in a send_message method. A message is sent on one channel only with two parameters, a RTCE page file name and the changes. Since the channel token is based on the page file name, this allows for pushing changes to only the client in the modified RTCE page.

Callback functions are defined on the client sides of the socket. In our system, the onmessage method (resp. called when receiving a message) is used. Onmessage receives the changed HTML and performs modifications during RTCE on the part of HTML.

B. Element for Communication

In a real-time collaborate editing system, each client’s individual part for communication is necessary and the treatment of it can be troublesome. Then, we note a specific technique for this matter in the following. In the case of Channel API, it is needed to load the code including class, method, and processes for communication in the client side. Figure 4 shows a <iframe> element that the code generates for communication. The <iframe> element makes a request to “http://talkgadget.google.com/talgadget/d” in a send_message method. A message is sent on one channel only with two parameters, a RTCE page file name and the changes. Since the channel token is based on the page file name, this allows for pushing changes to only the client in the modified RTCE page.

Push operations from the server are performed in a send_message method. A message is sent on one channel only with two parameters, a RTCE page file name and the changes. Since the channel token is based on the page file name, this allows for pushing changes to only the client in the modified RTCE page.

Push operations from the server are performed in a send_message method. A message is sent on one channel only with two parameters, a RTCE page file name and the changes. Since the channel token is based on the page file name, this allows for pushing changes to only the client in the modified RTCE page.
in which the token is specified as the parameter, and then the channel is established. For synchronous Web page editing, it is important to consider the treatment of the part that has static information for communication such as the `<iframe>` element.

At first, in our system, the client sends all of HTML to the server, and the HTML is pushed to the clients. The problem follows: when the RTCE page HTML is overwritten by the pushed HTML, each `<iframe>` element is unified into the `<iframe>` element in the editor’s RTCE page. Thus the communication does not go well. As we shall see later in the next chapter, performing synchronization in units of a changed element can solve this problem.

C. Differential Update Mechanism

By using all of HTML, our system originally synchronized the HTML of the RTCE pages, however several questions arise as follows. First one is, as described above, the problem that the `<iframe>` element for each client’s communication is overwritten with the editor’s one. To solve this, the technique excluding a static part from the synchronous subjects over RTCE is necessary. Another is a limitation push mechanism in a large size Web page due to a message size restriction, concretely 32Kbyte, on the send message method (in Figure 3). To avoid a bug, it is needed to reduce data. In addition to this, the data reduction is effective for reducing the synchronization gap.

In this section, we introduce a differential updates mechanism in which the synchronization of the web page and the changes of editing are performed in units of a changed element. In the present system, a static part is separated from editing part and a data for synchronization is reduced effectively.

We can represent the outline of our synchronization system in a simple diagram as Figure 5. In the system, each element in the RTCE page is confirmed using the selector, which is described in the form of Selectors API [6]. To obtain the selector, it is necessary to scan the DOM tree. Concretely, in our system, while tracing the HTML DOM tree from the target element to the `<body>` element, the current element’s order in a parent element’s children is recorded. Then, using a nth-child CSS selector, the selector is generated.

![Figure 5. Push System](image)

When a user performs a modification, the client sends the edit flag, the contents of the changed element, its selector and the URL of RTCE page to the page reflection module in the Figure 2. The module uses an ID, which is a combination of the RTCE page’s URL and the selector of the modified part, as a key name in the data store. In consequence, each change is managed uniquely. The edit flag indicates a modifying feature the editor used, and is necessary information for branch of synchronous process in the client side. The changed element’s contents include its style attribute and innerHTML.

![Figure 6. Differential Data Store](image)

In the client side, the HTML is overwritten using received data, and the clients naturally hold the modified content in common.

<table>
<thead>
<tr>
<th>ID/Name</th>
<th>edit_type</th>
<th>diff_contents (encoded)</th>
<th>page_id</th>
<th>style_attr</th>
<th>target_selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>name=95973b0-87e4-46fb-9e65-436b377dd7a.html - BODY&gt;DIV:nth-child(1)-DIV:nth-child(3)</td>
<td>editComment</td>
<td>comment test</td>
<td>.95973b0-87e4-46fb-9e65-436b377dd7a.html</td>
<td>position: absolute; color: black; background-color:transparent; z-index: 10; left: 472px; top: 400px;</td>
<td>BODY-DIV-nth-child(1)-DIV-nth-child(3)</td>
</tr>
<tr>
<td>name=95973b0-87e4-46fb-9e65-436b377dd7a.html - BODY&gt;DIV-nth-child(1)-DIV-nth-child(3)-DIV-nth-child(4)-DIV-nth-child(1)-DIV-nth-child(5)-DIV-nth-child(1)-DIV-nth-child(1)-DIV-nth-child(1)-P-nth-child(2)</td>
<td>editText</td>
<td>%a7f5973b0-87e4-46fb-9e65-436b377dd7a.html</td>
<td>/f3ad98b6-4b31-420a-bcb6-e6f1b66cd7b.html</td>
<td>null</td>
<td>BODY-DIV-nth-child(1)-DIV-nth-child(3)-DIV-nth-child(4)-DIV-nth-child(5)-DIV-nth-child(1)-DIV-nth-child(4)-DIV-nth-child(5)-DIV-nth-child(1)</td>
</tr>
</tbody>
</table>
When connecting to the RTCE page, the client makes a request to the page diff fetch module and the HTML changes of the RTCE page are returned. The changes are applied to the RTCE HTML thereby editing history are reflected. Besides this, during synchronous editing, the client overwrites HTML using the pushed elements. We greatly classify these reflection processes of a modification by three types of editing as follows.

First, when modifying a unique element in HTML such as changing the \textless title\textgreater element and altering the background color, the edit flag is set to \textit{editUniqueElement} and a target element is sent to the server. Then, in the receiving client, only the element is used to reflect a modification.

When text editing occurs, in the second place, the editor’s client sends \textit{editText} as the edit flag, a changed element and its selector to the server. Hence, the target element is retrieved using the selector and overwritten with the received element.

Furthermore, we describe the modifying comment as the last type of editing. The comment in the RTCE page is formed from a \textless div\textgreater element that has the style attribute giving a definition of its size and location. Since the modification of comment requires changing text, plus element insertion, movements and removal in a Web page, the synchronization technology of the comments is of particular importance. The text modification is performed in the same way as the ordinary text editing on the elements excluding the comments. The edit flag is set to \textit{insertComment} when creating the comment. Also, the \textless div\textgreater element is created as the comment and its innerHTML and style attribute are sent to the server. By using pushed data, the receiving clients newly generate a \textless div\textgreater element. If a user drags the comment in RTCE page, the client assigns the edit flag to \textit{movedComment} and sends the comment’s style attribute and selector. Then only overwriting the style attribute enables the synchronization of the comment’s location in the clients’ RTCE pages. The last edit flag is \textit{removeComment}, in this instance, it is needed to update the selectors of all comments in the data store as the mode changes the DOM tree structure. Although the \textit{insertComment} flag also changes the DOM tree by creating the \textless div\textgreater element, the element is added at the end of the \textless body\textgreater element, and then the selectors of other elements are immune to the insertion of comment. On the other hand, in case of removing the comment element, there is a possibility that a \textless div\textgreater comment element’s order in the \textless body\textgreater element is changed and its selector does not function. To solve this problem, in the comment removal mode, the selectors of all comments before and after editing are sent to the server and the server update data store.

IV. RTCE PAGE CREATION SYSTEM

This section describes the detail of our RTCE page creation system that can improve the accessibility of our system by using the bookmarklet, and moreover we mention usage of WFE-S and an example of how our system is used.

A. Creation Flow

Figure 7 shows RTCE page creation flow. The processing of the bookmarklet consists of two main steps. In the first one, the URL of the Web page being viewed is sent to the data store identification module on the server side, and the module confirms whether a RTCE version for the page exists. The returned result from the server is a RTCE page filename if a RTCE version exists in the data store, otherwise a \textit{notexist} status. When receiving the file name of RTCE version page, the client performs transition to the page. In case it did not, the process enters a RTCE page creation step detailed in follow.

On the client side, if a \textit{notexist} status message has been returned, a prompt asks for a password to be entered. The password is used to authentication for participating in a synchronous editing. If a password is set up correctly, the URL and the host name of the currently viewed Web page (which we will call the original page), its encoding, the requesting client’s user agent, and the password are sent to a RTCE page Creation module on the GAE server.

The module first creates a unique ID, which will be used as a file name for the RTCE page. Then, using the received URL, the module makes a HTTP request to retrieve the original page’s HTML. Since some original pages for a specific browser are provided, the user-agent parameter received before is used in the HTTP header.

In order to convert the original HTML in an editable one (a RTCE page), two types of tags are appended to the HTML. The first is the \textless base\textgreater tag. Since the RTCE page is hosted on the GAE server, it is necessary for relative paths in the original HTML to work. A \textless base\textgreater tag is added, thus, in the \textless head\textgreater tag of the page. The href attribute of this element is the original page’s host, which was received as a parameter at the beginning of the step. The second type of tags to be added is a \textless script\textgreater element that has functions to enable synchronous editing on the page. These elements include the libraries for using the Channel API on the client side, and the code to
communicate using the Channel API as well as to provide editing features.

The module inserts a new record, which consists of the original page’s URL, the HTML of the RTCE version one, and the password that was received at first, into the data store using the unique ID as key name. The encoding format at the RTCE page creation request is used for writing the RTCE page’s HTML record.

Finally, the RTCE page’s filename is returned to the client. On the client side, the page transitions to the RTCE page using this filename.

B. How to Use Our System

When a user activates the bookmarklet on a target Web page, the transition to the RTCE version page occurs. To begin a synchronous editing on the RTCE page, the user performs login process. A login prompt in Figure 8 can be accessed using a right click. Input of a password, which the creator set up in the RTCE page creation step, enables the user to edit the page. The password is sent to the authentication function on the server, and it is compared with a password fetched from the data store. Using status codes in HTTP, the server notifies whether the authentication is successful or not. In case of a match, the HTTP status code [200 OK] is returned to signal authentication success. If the password did not match, or if there is no record with such a file name, [401 Unauthorized] and [403 Forbidden] HTTP status codes are returned. When receiving the successful message, the codes required for editing functions are loaded, while reload is promoted for re-authentication.

Figure 9 shows a usage example of WFE-S. In a real-time collaborative editing, the users can use editing tools developed in previous research like text modifying, insert of comments, editing title, tables and lists. When a user makes a change, a RTCE page HTML is overwritten with the change. Then, the modified part of HTML and the selector indicating it are sent to the server, and the server pushes them to the clients. On the client side, the modified part is applied to the RTCE page HTML using the selector. Thus, each user can confirms changes in real-time.

V. APPLICATIONS

The techniques of our system make it easy to develop Web based applications. It is considered that our system can be applied to reducing development cost of Web application utilizing real-time. We have performed trial implementation of Web application as follows.

The implemented Web application is a simple notice board having a list of whereabouts, each user’s magnet and space for a message. It is used for the purpose of confirming the location of the users by the magnets location on the whereabouts list and noticing messages. We have developed this application using WFE-S; on the contrary, it is built with the HTML5 Canvas for the client’s interface and the WebSocket module of Node.js for push technology.

Figure 10 shows a notice board application we implemented. When using WFE-S, we create an HTML file including the element of message space and the <img> element of the whereabouts list as a sole source code file. Then, in the HTML page, we clicked the bookmarklet and the RTCE version of the page was generated. In the RTCE page, the magnets are created with the comments and moved freely. Also, the user can notice a message using text editing on the element for that purpose. The magnets movement and the message are synchronized, thus the users use the RTCE pages as a practical Web application.

Contrary to above, we face an important increase in development costs not using WFE-S. Firstly, it is needed to create a user interface from scratch. Using the HTML5 Canvas, we drew the magnets on a <canvas> element, and generated an object corresponding to each magnet. Moreover, to get the magnets draggable as the WFE-S comments, we added mouse events to the objects. We created a <textarea> element for the messages space and a <button> element sending a message to server. Secondly, the WebSocket server is necessary for push mechanism. We described the code of storing received data to a database and pushing it to the clients.
with Node.js. Thirdly, we are required to build the synchronization mechanism for magnets and messages. When a user drags a magnet, to synchronize the location of the magnet, the client sends coordinates of the target object to server, and the other clients transfer the object of the target magnet to received coordinates. The message space is synchronized in the same way.

It should be concluded, from what has been described above, that the development cost of a Web application can be reduced effectively by using our system. We counted the number of line of each source code. As a result, we obtained 39 lines of only the HTML file using WFE-S, against 755 with HTML5 and Node.js. Thus we see above conclusion is valid.

VI. EVALUATION AND DISCUSSION

We assume that physically dispersed users perform synchronous editing using our system. In order to prevent the collision of changes, it is required to minimize the synchronization gap in a real-time collaborative editing. In the following experiments, we measure the response to evaluate our system from the synchronization performance point of view. The response can be defined as the interval between the time when an HTML element modified by a user is sent to the server and the time when the changes are received and reflected in other clients. We define the synchronization performance by the gap in the response among clients. To assess the synchronization performance of our system, response measurement is carried out on the WFE system from previous research as well as the present research’s WFE-S.

A. Experimental Protocol

We base our experimental protocol on AJAX-related testing methods [7]. We assume that there are 5 concurrent users of the system, and that one client makes 100 changes to the page every 5 seconds. The response time for each of the four other clients is measured and logged. The clients are launched in a browser on the same machine, a desktop PC with a 2.7GHz Intel Core i5 CPU, and 4GB of RAM. To conduct experiments with the former system WFE, a server is needed. We use a netbook computer with a 1.7GHz Intel Core i5 CPU along and 4GB of RAM connected to the local network. Finally, server-client delay is inferred from the average ping of 1000 times requests. We obtained that a mean of 8.698ms and variance of 10.330 on the local network, and a mean of 45.091ms and variance of 4.348 for GAE.

To measure the response, starting and ending times must be acquired properly. In this experiment, this is performed on the clients and the times are considered synchronized because the clients are browsers on the same machine. We record the time when the client performing an edit (the editing client) sends a random change as the starting time. Also, we record the time when the four clients receiving changes (the receiving clients) finish overwriting HTML as the ending time. In the experiment using the previous research’s system, the editing client inserts a &lt;meta&gt; tag including the starting time into the changed HTML. The receiving clients extract the &lt;meta&gt; tag from the HTML retrieved by polling to obtain the starting time and subtract it from the measured ending time. The result is recorded as the experimental value on the browser’s console. In the experiment on WFE-S, the starting time is sent from the editing client to the server along with the changes. The receiving clients subtract the pushed starting time from the measured ending time and record the result on the browser’s console.

The standard deviation for the responses output in the four receiving clients is calculated over each change. Since the standard deviation represents the synchronization gap, observing this value enables to assess the synchronization performance.

B. Experimental Results

Figure 11 shows the distribution of the response obtained in 100 experimental measures. The horizontal axis shows the response time, while the vertical axis indicates the measured frequency of each response time. This figure shows that the response time is improved by using WFE-S. In fact, the average response time of WFE-S is 478.820ms, while the average of WFE’s response time is 632.940ms. The communication delay is 49.091ms on WFE-S and WFE has 8.698ms, thus the gap widens.

In addition, Figure 11 tells us that the response time of WFE is distributed over a wide range, whereas of WFE-S’s is concentrated in the vicinity of 300 to 600ms. Figure 12 shows this. In WFE, the standard deviation is in the neighborhood of 190 to 240, therefore it is hard to forecast the timing of synchronizing reflection during editing. On the other hand, in the case of WFE-S, the standard deviation is close to zero, the synchronization gap can be negligible and our system is acceptable for users consequently.

C. Discussion

In the previous research, clients were synchronized using polling. When performing collaborative editing, considering synchronization performance is important, however when trying to improve performance by reducing the polling interval, server load increases and impacts scalability.
To solve these problems, in the present system, two approaches are tested. First, GAE is used as base and synchronization is performed by push technology over the Channel API. The other one is synchronization technology using finite difference, i.e., the mechanism in which the changed element and its selector are sent and pushed to clients to synchronize Web pages. The results showed in Figure 11 demonstrate that using push technology is more effective for synchronization on clients when performing RTCE. The standard deviation of response time represents synchronization performance, which when improved results in less conflicting changes and therefore less loss of edited contents.

While we achieve some progress for the avoidance of editing conflicts in Real-Time Collaborative Editing, there is still room for improvement. One is accumulation of differential information. Since the client loads all modification history from server when transferring a RTCE page, accumulating the editing data considerably is inefficient. To solve this, on the server, we apply the changes to the RTCE page’s HTML on data store. The other is the problem of simultaneous modification conflicts by multiple users. We have improved the synchronization performance and reduce the chance of editing conflicts; however, when several users submit a change at close enough times, they can still occur.

A version control system can be a consistency resolution mechanism [8], which manages modifications and conflicts. However, the implementation of real-time version control system is difficult, and it is our future work.

VII. CONCLUSION

We have implemented WFE-S, a Real-Time Collaborative Editing system for existing Web pages in the cloud computing environment, that is Google App Engine. Our experiments showed that we could improve the synchronization performance of WFE by using push technology to perform synchronization and differential update mechanism. Our system can also hide underlying operations such as copy and transfer to the servers, making usage seamless by using the bookmarklet. This results in that the RTCE system for existing Web page becomes accessible to most users. Furthermore, it is expected that using the infrastructure of GAE largely improve scalability. We will evaluate the scalability of our system.

WFE-S enables us to make it easy to develop collaborative work systems as Web applications. We described the trial implementation of a collaborative Web application based on our system. It is considered that our system can be applied to reducing development cost of synchronous Web applications.

REFERENCES