# Differential Synchronization Mechanism for a Real-Time Collaborative Web Page Editing System WFE-S

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

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

***Abstract*—We propose a differential synchronization mechanism for Real-Time Collaborative Editing. We are developing WFE, a system for Real-Time Collaborative Editing for existing Web pages. We need to improve the response time to reduce conflicts among multiple users. We develop a DOM tree based differential updating mechanism to minimize update information for real-time editing. We implement the method on Google App Engine. We present comparison experiments that show our method can achieve practical this.**

***Keywords-component; real time collaborative editting; web application; web push mechanism; differential synchronization***

## I. INTRODUCTION

We propose a differential synchronization mechanism for Real-Time Collaborative Editing (RTCE) for existing Web pages. Research on collaborative editing using the Web is trending, and this field is growing importantly [1,2]. Our past and present research aims to enable several users to perform simultaneous modifications on existing Web pages in their browsers, while modified contents are reflected to other users in real-time.

There are various researches ongoing about differential synchronization in Web services [3], and our research as well focuses on realizing effective replacement of HTML information deltas in a Web browser. In this paper, we present our synchronous editing system making use only of information deltas as well as its implementation. In WFE, there is a chance for inconsistency to temporarily occur in the last version of the HTML file viewed by each user. In this paper, we show how we addressed the challenges encountered with WFE, namely our implementation for guaranteeing better consistency.

The paper is organized as follows. In the first section, we will introduce the challenges encountered when implementing a RTCE system for Web pages, which we tackle with our implementation. In Section II, we show limitations in the preceding research. In Section III and VI, we propose a DOM tree based differential synchronization mechanism and its implementation. Section V gives details on how to run the system. System evaluation is performed in Section VI. We then discuss and conclude this research in Sections VII.

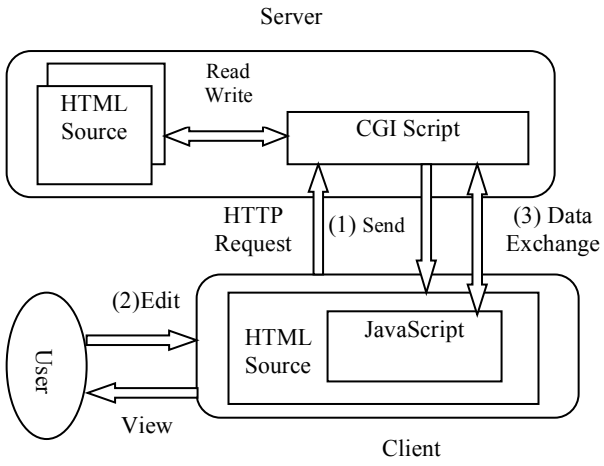


Figure 1. Workflow of WFE

## II. THE WFE SYSTEM FOR A REAL-TIME COLLABORATIVE WEB PAGE EDITING

We introduce the basic architecture, operations and challenges of WFE, a synchronous editing system for Web pages. This system constitutes the base of our research. By selecting text on a Web page open in a browser, users can edit the HTML content of the file [4,5]. This is done using the system's interface. There are several advantages to such a system. First, since the editing can be performed in the browser, there is no need to preview the results elsewhere, making editing fairly easy. Then, because the system provides an interface for editing, there is no need for knowledge of HTML or knowledge of its existence. Finally, because there is no need to upload data to the server, this represents an advantage in terms of burden on the user.

### *A. Architecture and Behavior of WFE*

Figure 1 outlines the basic architecture of WFE. As shown, it is mainly composed of two elements. The first is a CGI server script that provides a editable Web page and manages the associated HTML file. The second is a Javascript module that provides with the actual editing interface

and communicates edits to the server. The system is made possible by this communication between the CGI script on the server side and the Javascript incorporated in the editable HTML. There is also a necessity for the server that provides such an editable page to incorporate the required Javascript for editing.

However, there is no function for doing this in WFE automatically, and this increases burdening of the end user. We explain the behavior of WFE and detail the 5 steps required to operate it:

(1) Respond to a client HTTP request by returning an HTML file containing the Javascript for editing.
(2) On the client side, perform editing of the HTML file by using the UI provided by this Javascript.
(3) Send the edition output (as an HTML file) from client to server.
(4) On the server, use the payload sent in (3) to change the contents of the original HTML file sent in (1).
(5) Repeat steps 2 to 4.

This constitutes the basic workflow of Web page editing in WFE. Since WFE is made for synchronous editing of one single file, there is a need for one or more consistency maintenance mechanisms. However, when conflicts occur between changes after editing, this consistency is maintained by overwriting with the latter of the edits. This is problematic because earlier edits in such conflicts get erased and lost. Also, to avoid such conflicts it is desirable to notify other users when one of them has made a change. To this end, clients performing synchronous edition use a long polling mechanism to accomplish this in a matter of seconds, thus solving this issue to a certain extent.

### *B. Guaranteeing Consistency*

We introduce the challenges faced in WFE that constitute the subject of this paper. When conducting real-time synchronous editing in WFE, there is a chance for inconsistency to temporarily occur in the last version of the HTML file viewed by each user. We outline a scenario in which this type of inconsistency can occur. First, in the initial state, clients defined as A and B are able to modify one same file. In this state, A first performs a modification and sends its change to the server. In the scenario assumed by WFE, after this, the server updates the HTML file stored before the edit was made by A. Then, using B's long polling, it reflects the edits on the HTML being edited by B. However, we must consider the case in which B has sent edits to the server before its HTML could be updated by polling. In this case, the contents modified by A are not present in the payload sent by B to the server. Because of this, when the edits sent by A reach the server and the HTML file is overwritten, the latter edits by B which do not contain A's then overwrite the file and the contents edited by A are thus lost. We detail this scenario in Figure 2. Moreover, when this type of scenario takes place, from B's viewpoint there was no knowledge of A's modification. From the server as well, the only available information is that B's change occurred right after A's, making its version of the HTML file the final one, as if no problem ever occurred. Therefore, when A reacts to B's change by modifying the contents once again, as a result from every point of view there will be no consistency problem in appearance. However, when A is forced to perform the same edits two times, it can hardly be said that the system is easy to use.

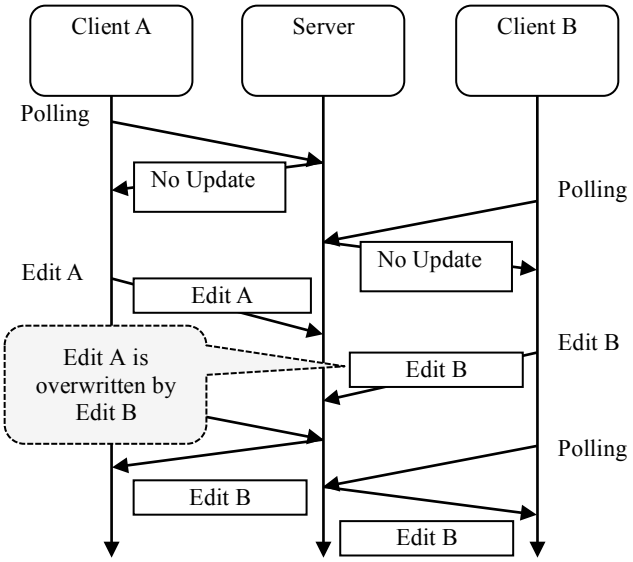


Figure 2. Inconsistent Situtation

## III. DIFFERENTIAL SYNCHRONIZATION MECHANISM

In this section, we will detail how we addressed the challenges encountered with WFE, namely our implementation for guaranteeing better consistency. We present a study of improvement methods for the obstacles introduced in Section II.B. There are mainly two approaches to the question. The first one is to ensure that the last HTML data is always the one used when making a change. The second is to ensure that even if the HTML edited was not the last version, the performed edits are consistent. The first approach thus consists in avoiding conflict as a whole, while the second acknowledges conflicts but solves them whenever possible. In the latter approach, one of the means available to solve the problem is to improve the composition of the HTML payload in order to perform better consistency maintenance. In this work, to implement this, we made it so that the edits sent by the client after editing are the modified contents and their position in the area of the page. As for implementing the first approach, we considered porting the system to Google App Engine to make use of push technology, and we give details of this implementation in Section IV.

In order to guarantee consistency when editing, our method in this work consists in using differential synchronization to avoid conflicts between edits. Even when conducting synchronous editing with multiple users, if the edits don't occur in the same areas of the pages it becomes possible to

reflect both of them during synchronization. Our system updates the HTML by using the position of DOM elements and by this seeks to improve performance when maintaining consistency. Several challenges are addressed in this paper when using this form of differential information to achieve real-time synchronization of the Web page between clients: 1) the method to acquire the deltas during edition, 2) how to manage these parts, and 3) how to reflect them. The following sections detail the respective solutions proposed.

### *A. Acquisition of Differential Information*

When editing a Web page in WFE, the system will first detect the modified elements' DOM among its HTML tags. Then, by overwriting the inner HTML of these elements, the HTML file is actually modified. In this work, to implement incremental updates, HTML of the target DOM elements as well as their XPath constitute the information deltas that the system acquires when a user edits the page. XPath is a syntax language used to specify the location of specific parts in an XML source [6]. By using XPath to locate edited parts of the Web page in WFE, other clients can get knowledge of where edition occurred. To send changes to the server after editing, in this research we will use these information deltas instead of the whole HTML source. Figure 2 shows differential information that makes use of XPath.

### *B. Management of Differential Information*

The differential information sent by the client to the server in the form of edits is accumulated on the server in an information delta storage module. When a client access takes place, this differential data gets appropriately dispatched if needed. Also, since in this method the differential data piles up with the number of edits, processing efficiency is lowered and the incremental update mechanism can fail. Thus there is a need to create a snapshot of HTML data at a proper timing, containing all the differential data up to a certain point, and update the HTML itself.

### *C. Application of Differential Information*

There are two cases to consider when reflecting the differential information. The first is when a new client accesses the Web page, the second is when there was an edit operation from a client already accessing the page. In the first case, when a new client accesses the Web page, HTML data of the Web page is read, but without any of the undergone edits applied to it yet. Therefore, all information deltas from the editing operations up until this moment are read at once, and sent to the client. Last state of the HTML data can be obtained by applying all differential data in chronological order from oldest to newest. The latest HTML data thus acquired is also sent to the server as a snapshot. We now describe the application of differential information when editing operations are being performed. When multiple clients are accessing the same Web page, if one has edited the page, the edit must be reflected on other clients as well. In this case, all the differential data is not required, only the data from the specific edit performed at this time. Therefore, when an edit occurs, the differential data is written to the differential datastore, but nothing more is read out from it the information delta is transmitted directly to every client. Figure 3. outline the flow of HTML data and differential data in each case.

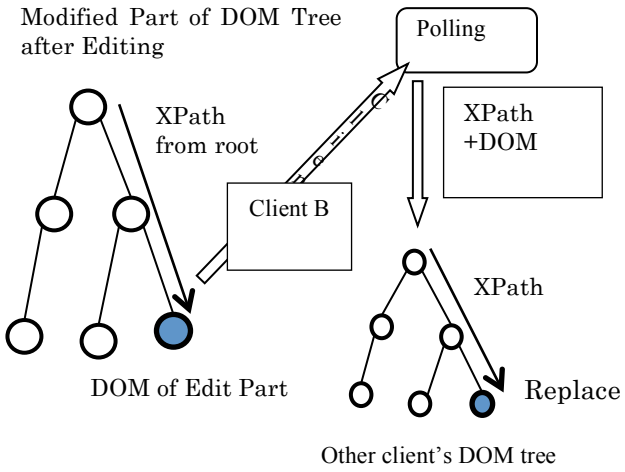


Figure 3. Incremental Updates with XPath

## IV. MPLEMENTATION OF PUSH TECHNOLOGY FOR DIFFERENTIAL INFORMATION

We call GAE-WFE the Google App Engine (GAE) reimplementation of WFE. In this section, we describe the changes to the existing WFE that were required when porting its architecture and behavior to GAE. Differences between GAE-WFE and WFE are the following. Since in both only the server-side environment changes, there is no great modification to the overall architecture. However, when using Google's infrastructure instead of a dedicated server, there is no possibility to use a CGI script to create, change or remove static files, among other limitations including CPU resources. In addition, in GAE-WFE, a new mechanism to carry out the edit detection is implemented using push technology with the Google App Engine API, instead of the long polling used so far.

### *A. Storage of HTML Data*

The original WFE managed Web pages by using a server-side Perl script to access, write to and create HTML files on the server's file system. However, since Google App Engine prohibits use of static files, a new mechanism to manage HTML data and differential data was needed. In the new GAE-WFE, we make use of the data store provided by GAE to manage storage and Web page update operations. The data store is a storage system provided by GAE to applications by means of an interface, GQL, that resembles SQL. Figure 4 shows an overview of the management mechanism of HTML data.

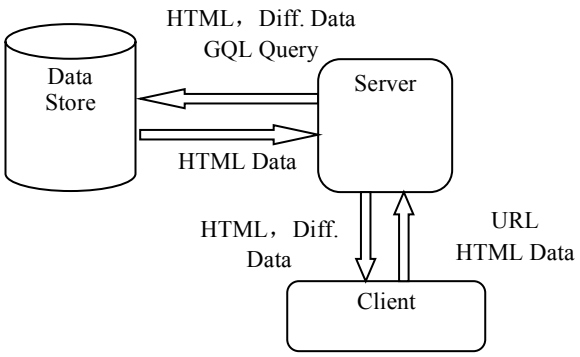


Figure 4. Handling Differential Data

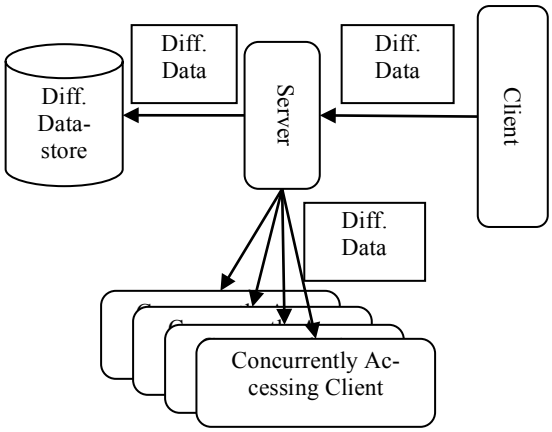


Figure 5. Edit Detection using Differential Information

*B. Registration of Target Web Pages for Edition*

We first describe how target Web pages that will be edited are registered to the system. A JavaScript bookmarklet is used to perform this task, which undergoes the following steps. First, the user opens in a browser the Web page he seeks to edit, and launches the bookmarklet. The bookmarklet accesses GAE-WFE and reads a registration script. This script then uses the page's URL as a key to consult the GAE-WFE data store for existence of the page. The server then returns a response indicating existence or not of the page's HTML data in the data store, with an ID if it exists. In the case the HTML data existed, the client uses the ID to access this HTML data, otherwise sends the HTML data of the page opened.

*C. Access to HTML Data*

When the client receives the ID of the HTML data and sends a request for it, the server responds by undergoing the following steps. First, it reads the URL to determine the file type according to the extension at the end of its path. By doing so, it can read the appropriate data from the data store accordingly. As a result, it is used in the content type header of the data to be sent back to the client.

*D. Edition of HTML Data*

When editing data, as mentioned above, the processing on the client side is different from existing ones: the differential data is passed to the client side and edition is performed there. In this section, we describe the workflow when a snapshot is sent to the server. HTML data from the edit received on the server side is processed in the following steps:

(1) The server receives the ID of the Web page under edition, as well as HTML contents of the edit.
(2) Remove DOM elements added for performing editing with JavaScript that were not in the original HTML data.
(3) Overwrite the HTML data contained in the data store with the received edit.

With this process, the client editing that used to replace HTML data on the server has been re-implemented with respect to GAE.

*E. Push Delivery Mechanism*

In the Google App Engine version of WFE, the push delivery mechanism is implemented by using the Comet-based Channel API provided by GAE. The Channel API create a sustainable connection between the application and Google servers, so that messages can be sent to the JavaScript on the clients, therefore making it possible to create a real-time connection not depending on polling. To implement push delivery, the following steps are necessary:

(1) Opening a socket of channel API for push delivery.
(2) Carrying out push delivery from the server to the client.
(3) Processing of the information thus received from the server.

Each operation is incorporated in the workflow of GAE-WFE. More specifically, in our implementation, (1) is conducted when requesting a Web page from the server, (2) is conducted after the server receives edits from the client that are collected in the data store, and (3) is conducted when the socket receives a messages from the client.

## V. IMPLEMENTATION OF SYNCHRONOUS EDITING

In the previous sections, we have detailed an implementation to improve the consistency performance of a Web synchronous editing system using push delivery. Although the present section does not relate directly to the system's capabilities, we present the implementation of a system-level necessary editor user management system, as well as solutions to address challenges encountered in the original WFE. There are several additions from the original WFE to the extended version on which the GAE version of WFE is based. Namely, embedding of links to other pages, edition of

the title and background, as well as login restrictions for editing. In the following we describe the changes necessary in this work on the GAE version. The GAE version uses the Channel API of GAE[7] for push. There are mainly three changes implemented:

(1) Adding various editing functions
(2) Editing restrictions by password mechanism
(3) Changing the script loading workflow

(1) Supported table and list editing, users can add and remove table rows, table columns, and list items. (2) Implemented a simple authentication mechanism. (3) Optimized the JavaScript loading strategy to speed up.

## VI. Evaluation Experiments

Finally, we conducted experiments to evaluate this new implementation of WFE. The measured performance is real-time reactivity. In this research, it is measured with the following metric: when several users are editing the same Web page, if one client performs a change, the time taken to propagate the changes to the HTML data in the browsers of other clients tends to be lower for real-time reactivity to be higher.

### A. Experiment Environment

We base our experimental protocol on AJAX-related testing methods [8]. The experiment environment is as follows. First, we assume that one client edits the page roughly once in every 30 seconds. This mean that if 1 edit was observed in 30s, there was 1 concurrent edit; if 1 edit was observed in 15s, there were 2 concurrent edits; if 1 edit was observed in 2s, there were 15 concurrent edits, and so forth. We also consider the target use of the system to be by small groups on the order of laboratory size, thus we expect it to withstand the concurrent editing of about 30 users. In other terms, the objective is for the server to be able of returning a response within a certain period of time given 1 edit request per second. Another objective is for the server to be able to deliver a response within the bounds of 1000ms after a request so that it can be considered of acceptable real-time reactivity and usable by multiple users synchronously. In addition, to examine the change in performance after improving our system, we conducted a similar experiment where the units used to propagate edits to other clients were whole pages instead of incremental blocks of differential information.

The experiment was carried out in the following way. Perform edits on the HTML data in random locations in a random time interval within the bounds of an access interval determined according to the number of clients. Here, record the start time just before sending the edit to the server. When the server receives the edit, push it to all the clients, and let them apply the edit. Then, record the end time as the time immediately after applying the edit. At this time, the response is measured by subtracting start time from end time. This process is repeated several times and the average response time is recorded.

| Concurrent Connections | Avg. Response Time Incremental Updates | Avg. Response Time Whole Page Updates |
|---|---|---|
| 3 | 796 | 1,861 |
| 6 | 705 | 7,435 |
| 9 | 715 | 8,385 |
| 12 | 2,551 | 9,955 |
| 15 | 2,082 | 10,063 |
| 18 | 3,253 | 10,148 |
| 21 | 3,348 | 10,123 |
| 24 | 5,066 | 11,719 |
| 27 | 2,192 | 10,667 |
| 30 | 3,673 | 11,168 |

TABLE I. Response Time by Number of Concurrent Users

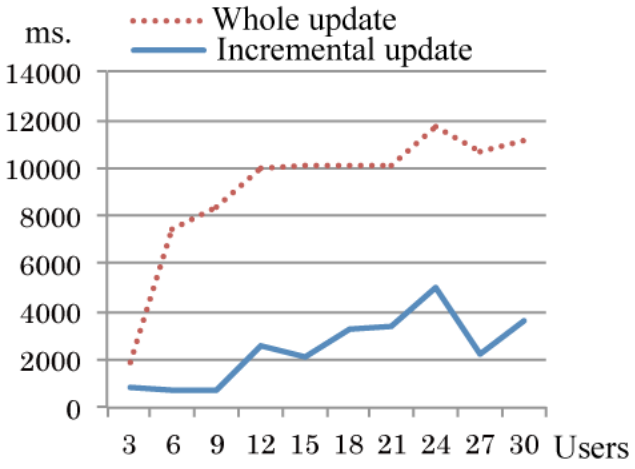


Figure 6. Response Time by Number of Concurrent Users

### B. Experimental Results

Table 1 and Figure 6 show the results of this experiment. According to the results, it is possible to satisfy the 1000ms boundary limitation when having up to 9 concurrent connections, but beyond that it becomes increasingly difficult. Additionally, the increase in response time is sudden between 9 and 12 concurrent connections but does not change significantly thereafter. The believed reason for this increase not to be a straight slope is the influence of random intervals over the number of simultaneous editors. Finally, when comparing the incremental update system with the whole-page update approach, we observe that the response time is reduced drastically.

### C. Discussion

First, we consider the experimental protocol. In this experiment, since it would be difficult to assess the real-time reactivity of the system with a fixed average in response time when several users are performing simultaneous edits, we

assumed a variation in the spacing of edit events. However, by looking at the results, when the number of users is increased it becomes difficult to predict the change in response time. Thus, we believe more control over the possible variations in the edit intervals is desirable.

The experiments were conducted in a group where every member was given the rigorous instructions to perform a random edit every 30 seconds. While the boundary limitation of 1000ms was satisfied only up to the order of 10 users, we witnessed significant improvement in real-time reactivity when comparing to the whole-page update approach. Before conducting the experiment, we expected the response time to soar with the number of concurrent connections, yet since this is not the case we believe there is opportunity for further investigation on the matter.

## VII. CONCLUSION

In this research, we have implemented a synchronous Web page editing system, WFE-S. The system makes use of incremental updates using the DOM tree and XPath to improve the efficiency of synchronous communication. We then confirmed by experiments that this improved the reactivity for updates. Though challenges still remain, this allows for maintaining consistency when performing synchronous editing among multiple users. By using this approach, the communication overhead could also be reduced, thus improving scalability. Our system enables several users to perform synchronous editing without knowledge of server management, making it possible to support collaborative work on existing Web pages.

## REFERENCES


[1] Matthias Heinrich, Franz Lehmann, Thomas Springer, Martin Gaedke, "Exploiting single-user web applications for shared editing: a generic transformation approach", Proceedings of the 21st international conference on World Wide Web, pp. 1057-1066, 2012.

[2] Max Goldman, Greg Little, Robert C. Miller, "Real-time collaborative coding in a web IDE", UIST '11 Proceedings of the 24th annual ACM symposium on User interface software and technology, 2011.

[3] Neil Fraser,"Differential synchronization", DocEng '09, 2009.

[4] Y.Fukagaya,T.Ozono,T.Ito and T.Shintani,"MiSpider: A Continuous Agent on Web Pages," WWW2005, pp.1008-1009, 2005.

[5] Nishi. K, et.al, "Implementing an Online Writable Web Page System and Its Applications", The transactions of the Institute of Electrical Engineers of Japan. C, A publication of Electronics, Information and System Society 125(4), pp. 660-665, 2005.

[6] Steve DeRose, "XPath 1.0, XML Path Language (XPath) Version 1.0, James Clark", http://www.w3.org/TR/xpath, 1999.

[7] Google, "ChannelAPI Overview (Python)", https://developers.google.com/appengine/docs/python/channel/overview, 2012/04/16.

[8] Engin Bozdag, Ali Meshah, Arie Van Deursen, "Performance testing of data delivery techniques for AJAX application", Journal Web Enginnering, Vol.8, Issue 4, 2009.