2015 29th International Conference on Advanced Information Networking and Applications Workshops

Implementation of a Collaborative Document Processing in the Cloud

Jiafei Wen, Xiaolong Wu, and Shui Lam
Dept. of Computer Engineering & Computer Science
California State University, Long Beach
Long Beach, CA, USA
(Jiafei.Wen, Xiaolong.Wu, Shui.Lam)@csulb.edu

***Abstract*—**Document processing is among one of the most widely used and well developed applications. With the recent fast development of high-speed internet and distributed computing, it becomes feasible to move document processing to web- and even cloud-based environment. The initial benefits of moving office documents into cloud for small and medium sized business are cost saving for purchasing, maintaining, and upgrading both software and hardware. However, the most significant advantage of this trend is to enable users of real-time collaborative editing on a shared cloud-based document. Therefore, moving office applications into cloud is an inevitable trend for the development of office application. A novel efficient document-processing model (DPC) in the cloud was proposed by the author in [12]. In this paper we implemented the DPC model in the Google cloud through the Google App Engine. Our preliminary cases testing results verified the proposed DPC model enabling users to process office document collaboratively with a proper granularity in the cloud.

***Keywords*—**Office document processing, cloud, collaborative editing.

I. INTRODUCTION

Today the office documents can be extremely sophisticated, which may contain complex formulas, graphic illustrations, video clips, and control information [6][7][8][9]. Meanwhile, the office document processing has evolved into large, complex, and powerful applications in order to reflect user requirements [1][3][11]. Among them, scientific paper is a good example where researchers usually need to develop and polish a technical report in a collaborative way. To accommodate such requirements, many different software systems have been developed by commercial and academic developers. Since a complete review of such systems is beyond the scope of this paper, we refer interested readers to [4][5].

Such requirements for collaborative teams could be much more difficult than those for standalone software [10]. As a result, the first problem is the costs incurred as for purchasing, maintaining, and upgrading required software and hardware. The second problem is accompanying system failures and human mistakes when the application becomes more complex. In order to satisfy the rapidly changing user demands and maintain the market, office document processing developers continue to upgrade their products by constantly adding new functions and features. These bring additional cost for people to upgrade their existing software and hardware to satisfy larger storage space and faster CPU demands. In another word, people will have to constantly invest more in order to support their routine document production and processing.

In our previous research [12], we have stated that cloud computing technology [2][13] can be an alternative approach to address this problem, since it enables services and storage facilities to be available and accessible over the Internet. Services provided by the cloud could be a web application, hence office document processing is a suitable candidate for it. With the web application of office document processing, users can create, edit, and share their documents without installing a complex software suite locally. Without doubt, user will save thousands of dollars on both hardware and software. Furthermore user can focus more on the creative work based on the latest document processing application, regardless of the cost of upgrading for software and hardware.

Google and Microsoft are two major companies among many others who provide office document processing as services in the cloud recently [14][15]. Details of these applications in the cloud will be introduced in the background section of this paper. However, in our previous research, we discussed that neither of them provides the collaboration with a proper granularity of collaborative editing on shared document. They both implemented the collaboration without dealing with different logical objects respectively. In this case, multiple users can edit one sentence, even one word, on the shared document concurrently, which will bring confusion and disorder among users in the past [14]. To address this problem, our previous research [12] proposed a Document Processing Model in the Cloud (DPC model). The DPC model enables users to process their office document collaboratively with a proper granularity in the cloud, which will be introduced briefly in the Section 3.

The main purpose of this paper is to describe the implementation of the proposed DPC model. By the implementation, office document processing will become a web application available in the cloud with a proper granularity of collaborative editing. Users are capable of handling their document processing work through browsers without installing the office document processing application on their own computer.

978-1-4799-1775-4/15 $31.00 © 2015 IEEE
DOI 10.1109/WAINA.2015.21

II. BACKGROUND

As mentioned before, Google and Microsoft offer office document processing through the cloud. Google Docs [14] is a free, web-based office suite, and data storage service offered by Google. It allows users to create and edit documents online while collaborating in real-time with other users. Microsoft Office 365 [15] is a commercial software plus services offering a set of products from Microsoft Corporation. Office 365 includes the Microsoft Office suites of desktop applications and hosted versions of Microsoft's Server products, delivered and accessed over the Internet. Both of these two applications claim to emphasize the support of collaboration of document editing among users. Through our testing and evaluation [12], we found that the collaborative editing of these two products do not have a proper granularity. For example, Google Docs enables multiple users to edit one shared document online, and it allows different users to edit one sentence, even one word, concurrently. However, in Google docs, even though users are notified where the change happened, they are not notified of the content changes since changes are not highlighted by Google docs.

As shown in Fig. 1, when a user is typing the word of "paragraph" in the second line, another user begins to type characters at the same place as the first user. In this case, the first user will be confused by the characters, since there is no visual difference between the characters he typed and the character typed by others. The first user only is noticed that there is another user editing this sentence but he does not know what the change is. This issue of confusion and disorder may become even worse when more users are working on a shared document. The reason for this issue is the editing operation does not base on different objects respectively. The content of the document is processed as one single object. We also found that the same problem exists in Microsoft office 365.

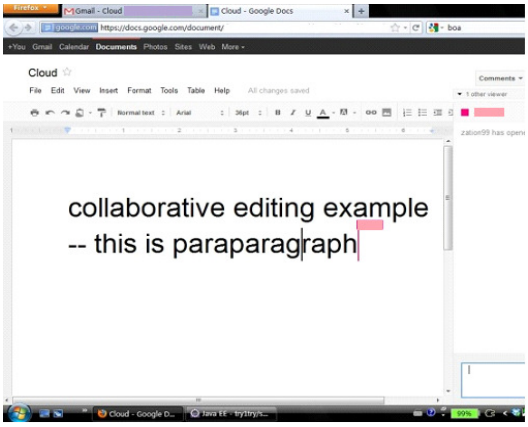

Figure 1. Collaborative editing in Google Docs.

To address this problem and enable users to process their office document collaboratively with a proper granularity in the cloud, our previous research proposed DPC model, which will be introduced briefly in the next section. Similar to Google Docs, the implementation of the DPC model described in this paper build a web application of it and deploy it as a "software as a service" in the cloud.

III. BACKGROUND

The DPC model was initially proposed by the authors back to 2010 [12] for the Document Processing in the Cloud. It is object-oriented based on XML logical structure [7][16]. It treats editable components in a document as distinct objects and it provides users respective access to every object. As a result, multiple users working on a shared document can perform collaborative editing based on distinct objects in real time. Such mechanism provides a more logical granularity for document processing collaboration since the DPC model processes the content of a document as logical objects rather than treating it as a string stream.

According to the DPC model, a whole document will be divided into thirteen objects listed in Table 1, including nine composites objects which contain basic object and four basic objects which are atomic.

TABLE I. DPC OBJECTS DESCRIPTION

	Object Name	Object Description
Composite Objects	Content	Specifies document's properties
	Meta data	Specifies meta data
	Header & Footer	Specifies headers and footers
	Style & Fonts	Specifies styles and fonts setting
	FootNote & EndNote	Specifies foot note and end note
	Comments	Specifies comments
	Paragraph	Specifies paragraphs' properties
	Table	Specifies tables
	Run	Specifies runs' properties of content in the parent field
Basic Objects	Hyperlink	Specifies hyperlinks
	Region	Specifies regions
	Text	Specifies literal text of runs displayed in the document
	Picture	Specifies pictures

Once been divided into DPC objects, the whole document becomes a unit of DPC objects. Each DPC objects is a unit of work distribution, which will be sent to processors in the cloud. The combination of all units is the entire document. User will be led to the target object in the cloud to finish their editing work. After all editing works finish, DPC objects will be collected and combined to form the final result document. In order to get a complete utilization in the cloud environment, DPC model also defined eight formulas as follow:

DPC = {DOC, MIDDLEWARE, PROCESSORS《bag PROCESSOR》}
DOC = {ROOT, ASSIST_INFO}
PROCESSOR = { APPS《bag APP》}
ROOT = {NONLEAF《bag COMPOSITE _OBJ》, LEAF《bag BASIC _OBJ》}
COMPOSITE_OBJ = {OBJECT, DESCENDANT《bag ROOT》}
BASIC_OBJ = {OBJECT}
OBJECT = {NAME, ACCESS_PATH, ON_EDITING}
ON_EDITING :: = Busy | Idle

The detailed information about these formulas is first introduced in [1]. It is worthwhile to note that in formula six, the DPC model use ACCESS_PATH described by XPath [17] to indicate the different objects after division and to lead users

to the target objects they want to edit. Fig. 2 shows DPC objects after division with its ACCESS_PATH in the cloud. Since the XPath of each node is unique in XML document, it can be identifiers of DPC objects in the cloud.

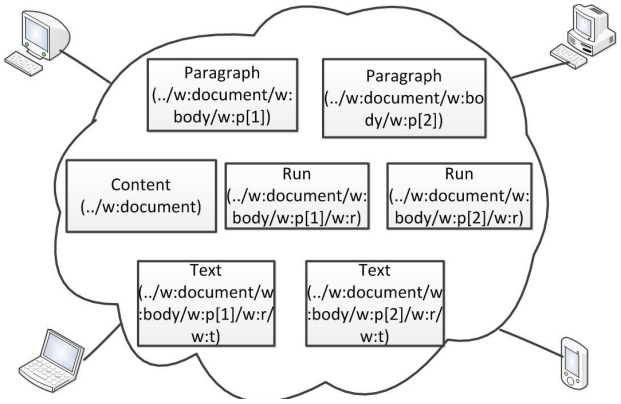

Figure 2. DPC objects with their ACCESS_PATH.

IV. IMPLEMENTATION

The web application of the DPC implementation is expected to offer document processing and to make the document accessible to authorized users from browsers. The owner of the document is able to invite others to work on the same document at the same time. The implementation is coded in Java for the backend and JavaScript for the frontend. For the frontend, we use the JQuery library [18] in JavaScript library to process basic text editing such as adding and deleting characters, changing the style and the size of characters, etc. Meanwhile, in order to provide better editing functions, we integrate the CKEditer [19] in our implementation. CKEditor is a text editor used inside web pages. Since our implementation is a web application for processing text on web page, the CKEditer is a good tool to accomplish this goal. Besides, CKEditer is licensed under flexible Open Source and commercial licenses [20].

As defined by formula one in the previous section, on the first level of the DPC model, there are three main components: DOC, Middleware, and Processors. Accordingly, there exist three corresponding main parts in the implementation represented in Fig. 3.

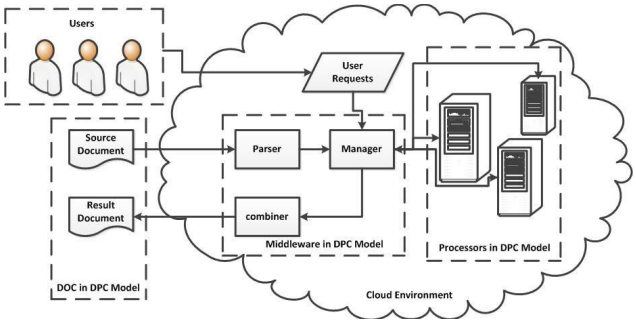

Figure 3. Overview of the implementation of the DPC Model.

As shown in Fig. 3, the DOC part includes the source document which will be uploaded to the cloud, and the result document which is downloaded from the cloud. In the cloud environment, middleware includes the parser, the manager, and the combiner. The workflows of the parser and the manager are shown in Fig. 4 and Fig. 5. The workflow of the combiner is to combine the results from manager according to the XPath of each result piece, which is similar to the reverse direction of the parser workflow.

As shown in Fig. 4, after receiving a source document, the parser records the XPath of each node in the source. Then, the parser divides the source document into pieces. After that the parser packs each piece with its corresponding XPath into DPC objects. Finally, the parser sends the DPC objects to the manager. We use ISO 29500 format document as an example to illustrate this process. An ISO 29500 document is described by several XML documents in Fig. 5, where docx format is based on ISO 29500, containing a "[Content_Types].xml" document and two file folders named "_rels" and "word" respectively. In the file folder named "word", there are several XML documents constituting the main content of the example document.

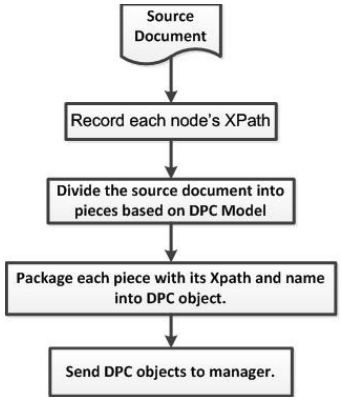

Figure 4. Main workflow of the parser.

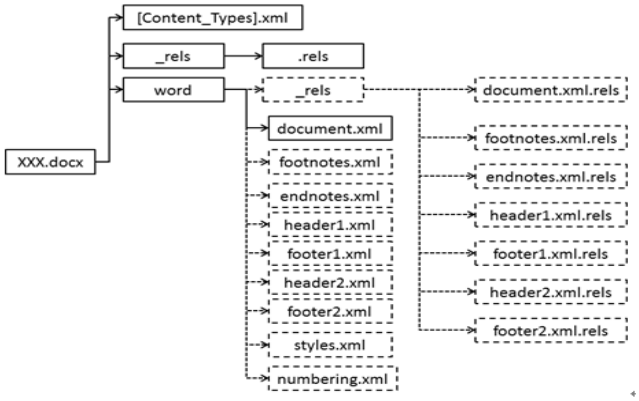

Figure 5. Structure of an ISO 29500 document.

In order to divide the example document to draw out DPC objects easily, the parser creates a new XML document to contain all of individual XML document from the source. With this it is easy to record the XPath of each DPC object

and it is also easy to keep the integrity of DPC objects. In the new XML for integrating all XML documents in the source document, the parser marks each XML document according to its own title, such as "workbook.xml", used to mark the piece from workbook.xml. The source document after this step of integration is shown in Fig. 6.

```
<xxx.docx>
  <[Content_Types].xml>
    <!--Here is the content of [Content_Types].xml-->
  </[Content_Types].xml>
  <_rels.xml>
    <!--Here is the content of _rels.xml-->
  </_rels.xml>
  <word.xml>
    <_rels.xml><!--Here is the content of _rels.xml--></_rels.xml>
    <document.xml><!--Here is the content of document.xml--></document.xml>
    <footnotes.xml><!--Here is the content of footnotes.xml--></footnotes.xml>
    <endnotes.xml><!--Here is the content of endnotes.xml--></endnotes.xml>
    <header1.xml><!--Here is the content of header1.xml--></header1.xml>
    <footer1.xml><!--Here is the content of footer1.xml--></footer1.xml>
    <header2.xml><!--Here is the content of header2.xml--></header2.xml>
    <footer2.xml><!--Here is the content of footer2.xml--></footer2.xml>
    <styles.xml><!--Here is the content of styles.xml--></styles.xml>
    <numbering.xml><!--Here is the content of numbering.xml--></numbering.xml>
  </word.xml>
</xxx.docx>
```

Figure 6. Source document after integration.

We use Extensible Stylesheet Language Transformations (XSLT) technology to integrate these individual documents. XSLT is a XML-based language used for the transformation of XML documents [21], to generate a new XML document without changing the original XML document. For example, Fig. 7 shows the main steps of the Stylesheet of XSLT for integration, through which all the XML documents in the source document are integrated into one XML document.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!--apply on workbook.xml-->
  <xsl:template name="spreadsheets" match="/">
  <!--add file name as mark-->
  <xxx.docx>
    <!--output [Content_Types].xml-->
    <[Content_Types].xml>
      <!--copy the content of [Content_Types].xml-->
      <xsl:copy-of select="document(xxx/[Content_Types].xml/>
    </[Content_Types].xml>
    <!--output _rels.xml-->
    <_rels.xml>
      <!--copy the content of _rels.xml-->
      <xsl:copy-of select="document(xxx/_rels.xml/>
    </_rels.xml>
    <!--Other nodes are similiar-->
  </xxx.docx>
</xsl:stylesheet>
```

Figure 7. XSLT Stylesheet for integration.

Through the result document of integration, the parser records each node's XPath. For example, the <document.xml> node's XPath is "xxx.docx/word.xml/document.xml". After recording the XPath, the parser divides the result document and then takes out the corresponding DPC objects. If a user wants to edit the content of a paragraph, he/she must have access to that object. Obviously, there will be some nodes which do not belong to any DPC object. These nodes will also be sent to the manager as backup. After receiving the data from the parser or user, the manager will call the corresponding java servlet to handle the data. The workflow of the manager is shown in Fig. 8.

As shown in Fig. 8 the manager processes two kinds of inputs. The first consists of user requests and the second consists of DPC objects. For the DPC objects, the manager will determine whether these DPC objects have been saved. If the input has not been saved, it means this document is uploaded into the cloud for the first time and the manager will save them in its storage. Then, manager will send the DPC objects to corresponding processors. If the input has been saved, it means those DPC objects come from processors rather than from the parser and the manager will refresh the DPC Objects saved before. DPC objects in storage are used to display the whole document to users, so they need to be refreshed on time.

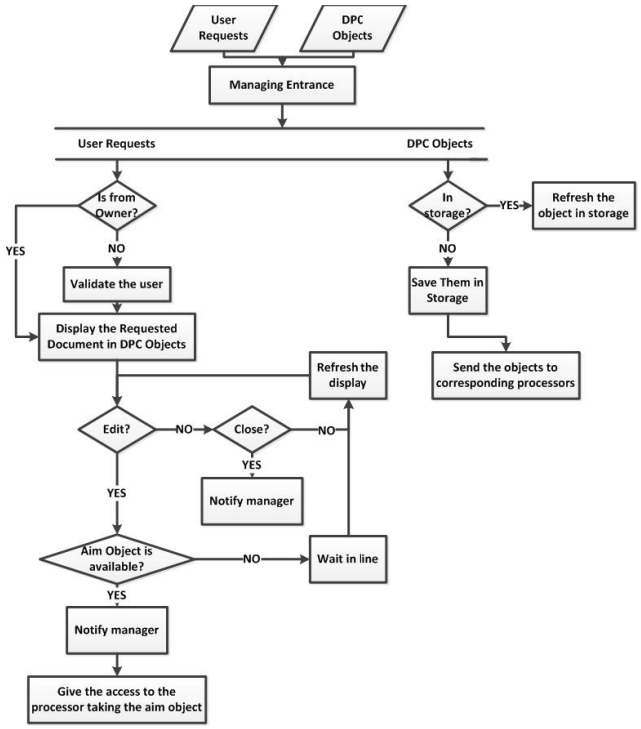

Figure 8. Workflow of the manager.

If the input consists of user requests, the manager will check whether these request comes from the owner of the target document or not. The owner of the document, who uploaded the document before, will decides which user can edit the document. In order to do that, the owner needs to send invitations to users who will be allowed for collaborative editing. Then users will have access for such editing once they log in. However, if it comes from the owner, the manager will display the whole document on the browser. If it is not, the manager will validate the user firstly, and then the browser can display the document after validation. On the other hand, if the user wants to edit the uploaded document, the request will indicate which part of the document the user wants to edit. By such request, the manager will check whether the

object in the document is available for editing or not. If it is available, the manager will authorize user to edit. Otherwise, the user needs to wait in the line for the target object. Since the document is saved on demand, the manager will refresh the display periodically.

The implementation of the DPC model in this paper is deployed in the Google Cloud through Google App Engine [22]. Google App Engine is a cloud computing platform providing platform as service and hosting web application in the Google-managed data centers. The application implemented the DPC model will be enabled by it as a web application hosted in the Google-managed data centers.

V. TESTING

In this section, we presented our designed test cases to test uploading, editing by both single user and collaborative editing by multiple users functions of the DPC model implementation. Four different test cases and their results are described in the following paragraphs respectively. These test cases run on the browser of Firefox 7.0 for functional testing of black box testing.

TABLE II. TEST CASE I

Type	Upload the example document
Purpose	Upload example document into cloud through implementation of the DPC and display its content on the browser
Test data	Docx document with two paragraphs
Test steps	Start the application of the implementation of the DPC by a browser; select the example document; click upload button
Test result	Pass
Screenshot	Shown in Fig. 9

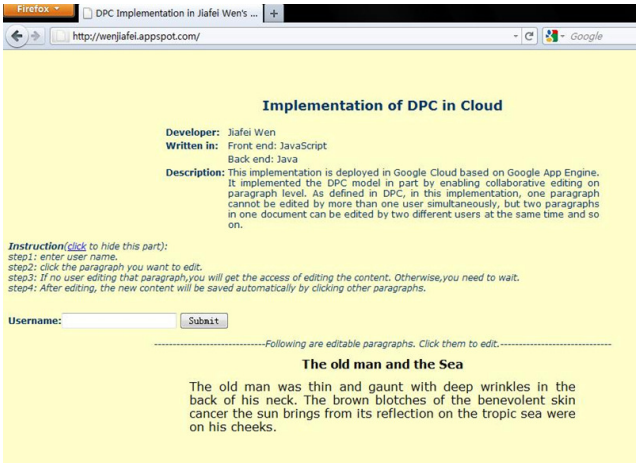

Figure 9. Screenshot of test case I.

As shown in Table II: Test Case I and Fig. 9, the paragraphs in black fonts are the content of the example document, and they are also editable paragraphs on the browser. On the other hand, users can edit them after login by clicking the content, as described in the Table III: Test Case II and Fig. 10. When a user clicks the paragraph, the paragraph will change to an editable field.

TABLE III. TEST CASE II

Type	Edit the example document on the browser
Purpose	Edit one paragraph of example doc displayed on browser
Test data	Example document uploaded by test case one
Test steps	Click the first paragraph and edit the content
Test result	Pass
Screenshot	Shown in Fig. 10

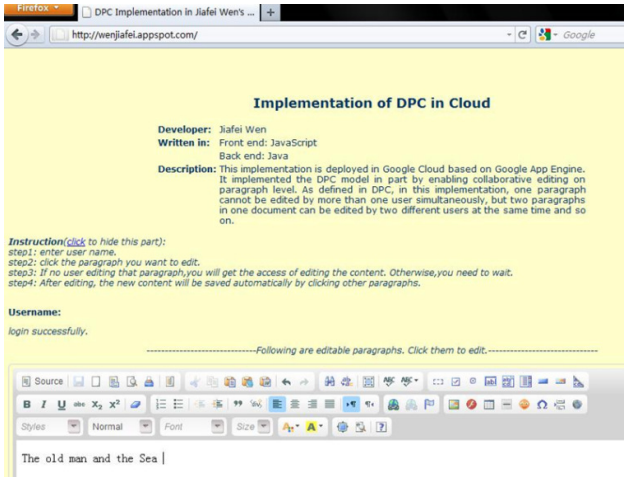

Figure 10. Screenshot of test case II.

When a paragraph is under editing, it cannot be edited by other users through other browsers. If a user clicks the paragraph which is under editing, he/she will receive the information saying "another user is editing this paragraph", as described in the Table IV: Test Case III and shown in Fig. 11.

TABLE IV. TEST CASE III

Type	Edit a paragraph which is under editing
Purpose	Edit a paragraph which is under editing by using different user name. The browser will give the invalid information to user
Test data	Example document in the cloud
Test steps	Start the application of the implementation of the DPC by a browser; select the example document; click upload button
Test result	Pass
Screenshot	Shown in Fig. 11

Multiple users edit different paragraphs on the shared document though different browsers at the same time are presented in Table V: Test Case IV and Fig. 12.

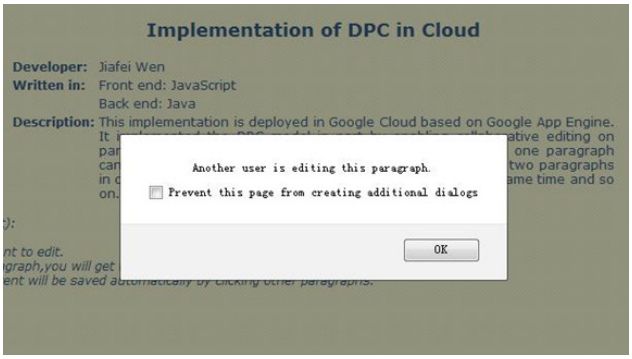

Figure 11. Screenshot of test case III.

TABLE V. TEST CASE IV

Type	Two users edit two paragraphs of one shared document through different browsers at same time
Purpose	Collaborative editing through different browsers
Test data	Example document in the cloud
Test steps	Login the application through two browsers by different user names; Click the first paragraph in one browser; Click the second paragraph in another browser. Two paragraphs can be edited at same time through different browser
Test result	Pass
Screenshot	Shown in Fig. 12

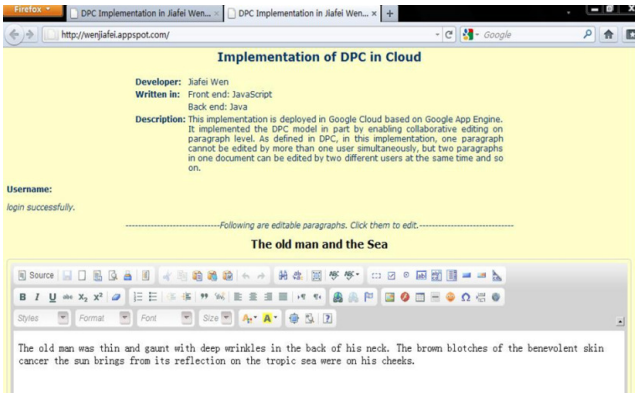

Figure 12. Collaborative editing through different browsers for case IV.

VI. CONCLUSION

Moving office document processing into the cloud is a recent trend in IT industry. It not only helps users save cost on their software and hardware upgrading, also enables user of collaborative editing on a shared document through internet. Our previous research has proposed a DPC model for efficient office document processing in the cloud. In this paper, we introduce the implementation of the DPC model, which is deployed in the Google cloud through Google App Engine. The implementation is coded in Java for the backend and JavaScript for the frontend. Our implementation proves that the web application of the DPC implementation offers document processing and makes the document accessible to authorized users from browsers. The owner of the document is able to invite others to work on the same document at the same time. Meanwhile, results from four different designed test cases further confirm that our proposed DPC model provides a proper granularity of collaborative editing.

REFERENCES

[1] J. C. Adler and S. Noël, "Evaluating and implementing a collaborative cffice document dystem," Interacting with Computers, vol. 18, issue 4, 2006, pp. 1-18.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brabdic, "Cloud computing and emerging IT platforms: vision, hype, and readily for delivering computing as the 5th utility," Future Generation Computer Systems, vol. 25, issue 6, June 2009, pp. 599-616.

[3] E. Kim and K. Severinson Eklundh, "How academics co-ordinate their documentation work and communicate about reviewing in collaborative writing," Report #TRITA-NA-P9815, IPLab-151, Royal Institute of Technology (KTH), Sweden.

[4] J. Newman and R. Newman, "Three modes of collaborative authoring in computers and writing: state of the art (P.O. Holt and N. Williams, Eds.)," Oxford: Intellect Books, 1992, pp. 20-28.

[5] S. Noël and J. M. Robert, "How the Web is Used to Support Collaborative Writing," Behaviour & Information Technology, vol. 22, 2003, pp. 245-262.

[6] S. Noël and J. M. Rober, "Empirical study on collaborative writing: what do co-authors do, use, and like?" Computer Supported Cooperative Work: The Journal of Collaborative Computing, vol. 13, 2004, pp. 63-89.

[7] S. Petride, A. Tarachandani, N. Agarwal, and S. Idicula, "Managing and processing office documents in oracle XML database," Proc. the 3rd International Conference on Advances, Knowledge, and Data Applications, 2011, pp. 89-95.

[8] I. R. Posner and R. M. Baecker, "How people writet together, in readings in groupware and computersSupported cooperative work: assisting human-human collaboration (R.M. Baecker, Ed.)," 1992, pp. 239-250.

[9] C. B. Rizzi, C. M. M. C. Alonso, E. B. Hassan, L. M. R. Tarouco, and L. M. J. De Seixas, "EquiText: a helping tool in the elaboration of collaborative texts," Proc. 2000 Society for Information Technology & Teacher Education International Conference, pp. 2314-2319.

[10] C. Sun and C. Ellis, "Operational transformation in real-time group editors: issues, algorithms, and achievements," Proc.the 1998 ACM conference on Computer supported cooperative work.

[11] Y. Tang, C. Yan, and C. Y. Suen, "Document processing for automatic knowledge acquisition," IEEE Transactions on Knowledge and Data Engineering, vol. 6, February 1994. pp. 3-21.

[12] J. Wen, S. Lam, and X. Wu, "A Model for office document processing and collaboration in the cloud," Proc.the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications, Jul. 2011, Las Vegas, USA.

[13] L. Zhang and Q. Zhou, "CCOA: cloud computing open architecture," Proc. IEEE International Conference on Web Services, 2009.

[14] http://www.google.com/google-d-s/intl/en/tour1.html.

[15] http://office365.microsoft.com/en-US/online-services.aspx.

[16] XML Path Language (XPath) 2.0. W3C Recommendation, January 2007.

[17] http://www.w3.org/TR/xpath20/.

[18] http://jquery.com/.

[19] http://ckeditor.com/what-is-ckeditor.

[20] http://ckeditor.com/license.

[21] XSL Transformations (XSLT) Version 1.0; http://www.w3.org/TR/xslt.

[22] http://en.wikipedia.org/wiki/Google_App_Engine.