Journal of
Computer Science & Systems Biology

Wen and Wu,, J Comput Sci Syst Biol 2014, 7:5
DOI: 10.4172/jcsb.1000153

Research Article Open Access

Implementation of a Collaborative Document Processing in the Cloud

Jiafei Wen and Xiaolong Wu*

Department of Computer Engineering and Computer Science, California State University Long Beach, USA

Abstract

Document processing is one of the most widely used and well developed With the recent fast development of high-speed internet and distributed computing, it is possible to move document processing to web-based, and even cloud-based. The initial benefits of moving office documents into cloud for small and medium sized are business cost saving for buying, maintaining, and upgrading both software and hardware. However, the most significant advantage of doing 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 is proposed. Detailed description and functioning of this model is briefly discussed first in this paper. Next, we implemented the DPC model in the Google cloud through the Google App Engine. Our cases testing verified the proposed DPC model enabling users to process their office document collaboratively by a proper granularity in cloud.

Keywords: Office document processing; Cloud; Collaborative editing

Introduction

Today the office documents can be extremely sophisticated, which may contain complex formulas, graphic illustrations, video clips, and control information [1-3]. Correspondingly, the office document processing has evolved into large, complex and powerful applications based on user requirements [4-6]. Among them, scientific paper is a good example, in which researchers need to jointly develop and refine a document in a collaborative way. To address the requirements of collaborative editing, many different software systems have been developed, some commercial and some academic. Since an exhaustive review of such systems is beyond the scope of this paper, we refer reads to read articles [7,8]. The requirements of collaborative teams can be much more difficult than those for standalone software.

As a result, the first problem is the costs that are incurred as for purchasing, maintaining, and upgrading required software and hardware. The second problem is more 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 upgrading their products by constantly adding new functions and features. These upgrades cost people on not only software but also hardware, since upgrades demand more storage space and faster CPU. Hence, people will have to continue to invest on both software and hardware in order to support their routine document production and processing.

In our previous research [9], we have stated that cloud computing technology [10] can be an alternative approach to address this problem, since it enables services and storage facilities to be provided over the Internet and allows users to access the services and storage facilities through the Internet. Services provided by the cloud can be a web application, and 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. In this case, user can expect to save thousands of dollars on both hardware and software. Besides, user also can put more focus on the creative work based on the latest document processing application, regardless of the cost of upgrading for software and hardware.

Until now, there are two companies, Google and Microsoft, provide office document processing as services in the cloud [5,7]. 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. To address this problem, our previous research 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 introduction.

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 the collaborative editing. Users are able to perform their document processing work through browsers without installing the office document processing application on their own computer.

Background

As mentioned before, Google and Microsoft offer office document processing through the cloud. Google Docs 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 is commercial software plus services offering a set of products from Microsoft Corporation. Office 365 includes the Microsoft Office suites of desktop applications and hosted

***Corresponding author:** Xiaolong Wu, Department of Computer Engineering and Computer Science, California State University, Long Beach, 1255 Bellflower Blvd, Long Beach, CA 90840, USA, Tel: 562-985-2910; Fax: 562-985-7823; E-mail: xiaolong.wu@csulb.edu

Received July 28, 2014; **Accepted** August 16, 2014; **Published** August 18, 2014

Citation: Wen J, Wu X (2014) Implementation of a Collaborative Document Processing in the Cloud. J Comput Sci Syst Biol 7: 174-179. doi:10.4172/jcsb.1000153

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, we consider 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 of any changes, since changes are not highlighted by Google docs.

As shown in Figure 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).

To address this problem and enable users to process their office document collaboratively by 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.

DPC Model

The DPC model is Document Processing in the Cloud model. It is object-oriented based on XML logical structure [10]. It treats editable components in a document as distinct objects, and it gives users respective access to each object. As a result, multiple users working on a shared document can do 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.

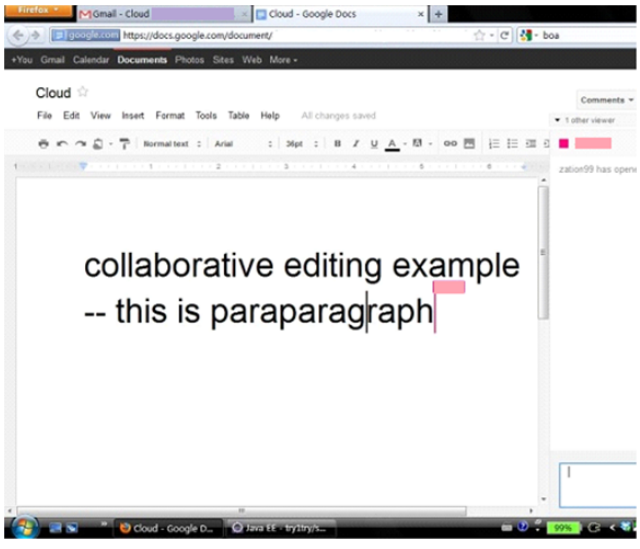

Figure 1: Collaborative Editing in Google Docs.

	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 which shall be displayed in the document
	Picture	Specifies pictures

Table 1: DPC Objects Description.

DPC={DOC, MIDDLEWARE, PROCESSORS《bag PROCESSOR》}	(1)
DOC={ROOT, ASSIST_INFO}	(2)
PROCESSOR = {APPS《bag APP》}	(3)
ROOT={NONLEAF《bag COMPOSITE _OBJ》, LEAF《bag BASIC _OBJ》}	(4)
COMPOSITE_OBJ={OBJECT, DESCENDANT《bag ROOT》}	(5)
BASIC_OBJ={OBJECT}	(6)
OBJECT={NAME, ACCESS_PATH, ON_EDITING}	(7)
ON_EDITING ::=Busy | Idle	(8)

Table 2: DPC model defined eight formulas.

Defined in the DPC model, firstly, a whole document will be divided into thirteen objects listed in Table 1, based on which the whole document is divided. DPC Objects includes nine composites object which include basic object, and four basic objects which are atomic. Each of the DPC objects cannot be edited by more than one user at any time (Table 1).

After being 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: (Table 2)

The detailed information about these formulas is introduced by [4]. It is worthwhile to note that in formula six, the DPC model use ACCESS_PATH, described by XPath [2], to indicate the different objects after division and to lead users to the target objects they want to edit. Figure 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).

Implementation

The web application of the DPC implementation offers document processing and makes the document accessible to authorized users form 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. In the frontend, we use the JQuery library [3] in JavaScript library to process the basic text editing, such as adding and deleting characters, changing the style and the size of characters, and so on. Meanwhile, in order to provide better editing functions, we integrate the CKEditer

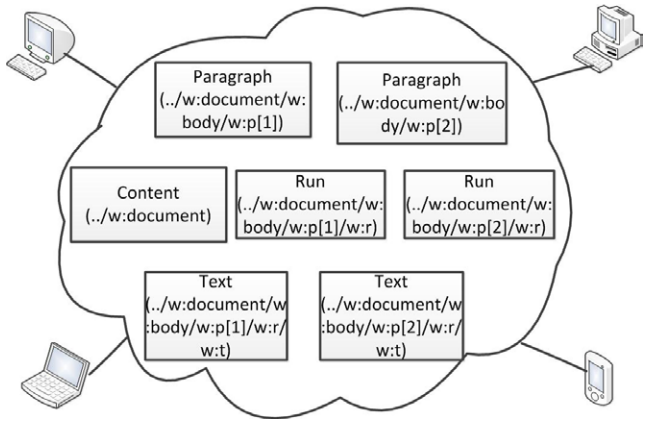

Figure 2: DPC Objects with Their ACCESS_PATH.

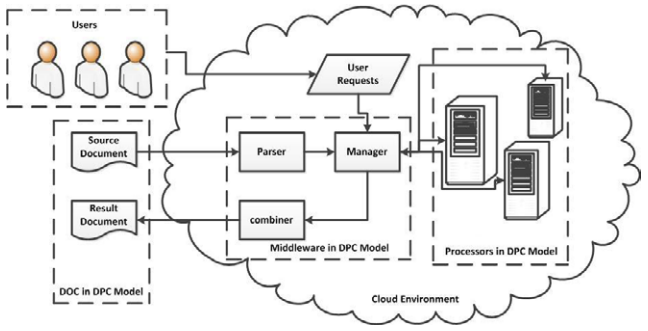

Figure 3: Overview of the Implementation of the DPC Model.

[11] 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 for us to accomplish this goal. Besides, CKEditer is licensed under flexible Open Source and commercial licenses [6], so that it can be integrated in our application legally.

As defined by formula one in the Chapter three, on the first level of the DPC model, there are three main components which are DOC, Middleware, and Processors. Accordingly, there are three corresponding main parts in the implementation (Figure 3).

As shown in Figure 3, the DOC part includes the source document which is going to 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 Figures 4 and 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 Figure 4, after receiving a source document, the parser records the XPath of each node in the source document. Then, the parser divides the source document into pieces based on the DPC model, after which 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, so actually it is a collection of XML documents (Figure 5). In Figure 5, docx format is based on ISO 29500, which contains 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.

In order to divide the example document to draw out DPC objects easily, the parser creates a new XML document to contain all of the individual XML documents in the source document. By doing so, it is easy to record the XPath of each DPC object, and it is also easy for keeping the integrity of DPC objects. In the new XML for integrating all XML documents in the source document, the parser marks each XML document by its own title, such as "workbook.xml", used to mark the piece from workbook.xml. After this step of integration, the source document is as shown in (Figure 6).

We use Extensible Style sheet Language Transformations (XSLT) technology to integrate these individual documents. XSLT is a XML-based language used for the transformation of XML documents [12], which is used to generate a new XML document without changing the original XML document it based on. For example, Figure 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. As mentioned before, after integration, the source document is as shown in Figure 6 as one single XML document.

Through the result document of integration, the parser records each node's XPath. For example, the <document.xml> node's XPath is

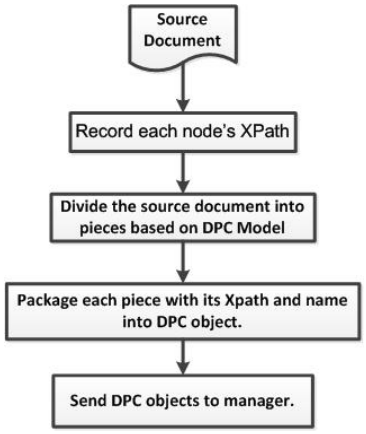

Figure 4: Main Workflow of the Parser.

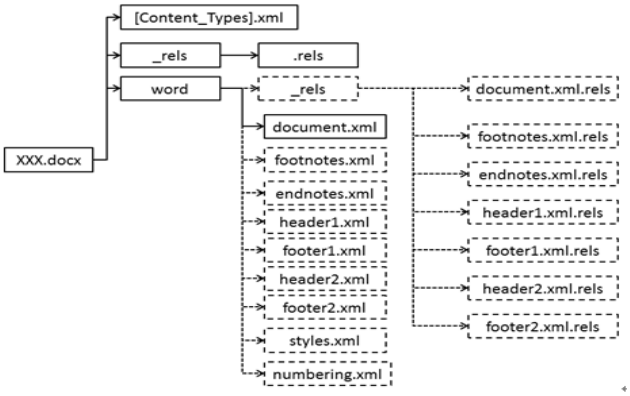

Figure 5: Structure of an ISO 29500 Document.

```
<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.

```
<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 Style sheet for Integration.

"xxx.docx/word.xml/document.xml".

After recording the XPath, the parser divides the result document based on the specification of DPC and then draws out the corresponding DPC objects. If a user wants to edit the content of a paragraph, he 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 Figure 8 [13-19].

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

If the input consists of user request, the manager will check whether the request is coming from the owner of the target document or not. The owner of the document, who uploaded the document, decides which user can edit the document. The owner needs to send invitations to users who will be allowed for collaborative editing. Then those users will be authorized for such editing once they log in. If it is coming 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 will display the document after validation. If the user wants to edit the uploaded document, the request will indicate which part in 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 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 [13]. 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 [20-22].

Testing

In this section, we designed test cases to test uploading, editing by a single user, and collaborative editing by multiple users functions of the DPC model implementation. Four main 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 (Figure 9).

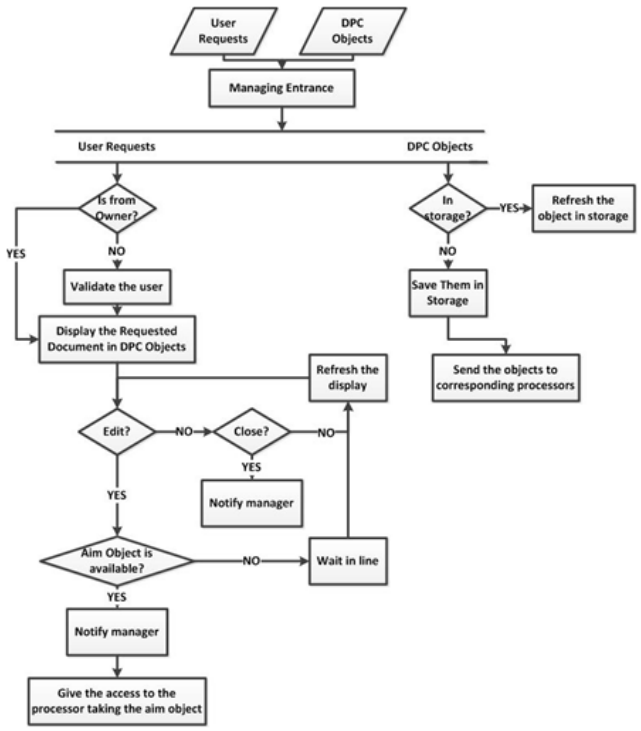

Figure 8: Workflow of the Manager.

Test Case I:	Upload the example document.
Purpose:	Upload the example document into the cloud through the 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 Figure 9

As tested in Test Case I, the paragraphs in black fonts are the content of the example document, and they are also editable paragraphs on the browser. Users can edit them after login by clicking the content, as described in the Test Case II. When a user clicks the paragraph, the paragraph will change to an editable field (Figure 10).

Test Case II:	Edit the example document on the browser.
Purpose:	Edit one paragraph of the example document displayed on browser.
Test data:	Example document uploaded by test case one.

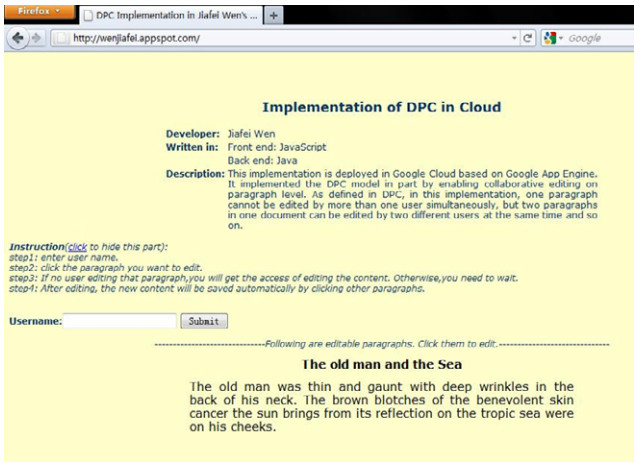

Figure 9: Screenshot of Test Case I.

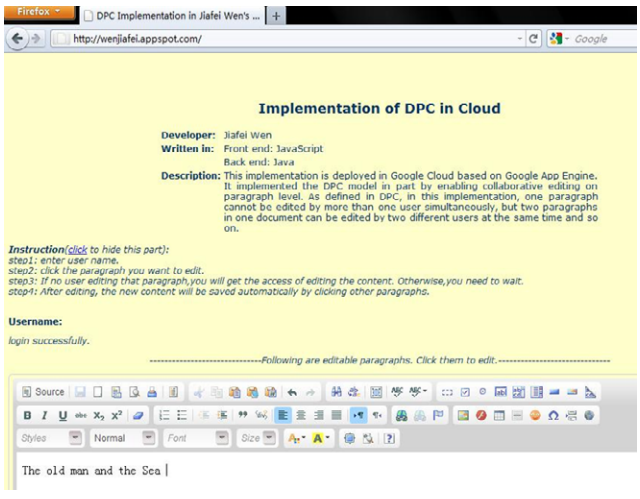

Figure 10: Screenshot of Test Case II.

Test steps:	Click the first paragraph and edit the content.
Test result:	Pass.
Screenshot:	Shown in Figure 10.

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 will receive the information saying "another user is editing this paragraph", as described in the Test Case III (Figures 11 and 12).

Test Case III:	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:	Two users login by two different user names through two different browsers; One user clicks the first paragraph, and then the other user also clicks the first paragraph.
Test result:	Pass.
Screenshot:	Shown in Figure 11.

Test Case IV:	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 Figure 12.

Conclusion

Moving office document processing into the cloud is a trend in IT industry, since it not only help user save cost on their software and hardware upgrading, but also enable user to do collaborative editing on a shared document through internet. Our previous research has proposed the 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 works well and the results confirm that

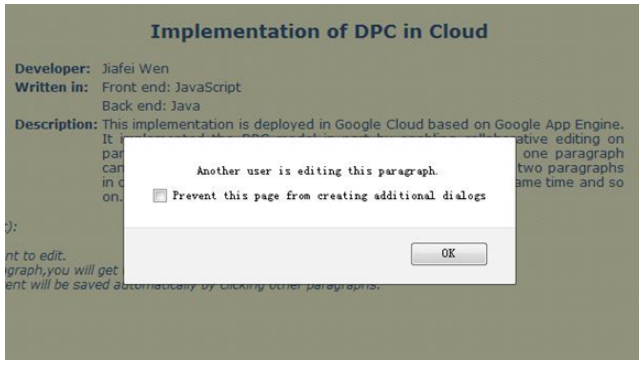

Figure 11: Screenshot of Test Case III.

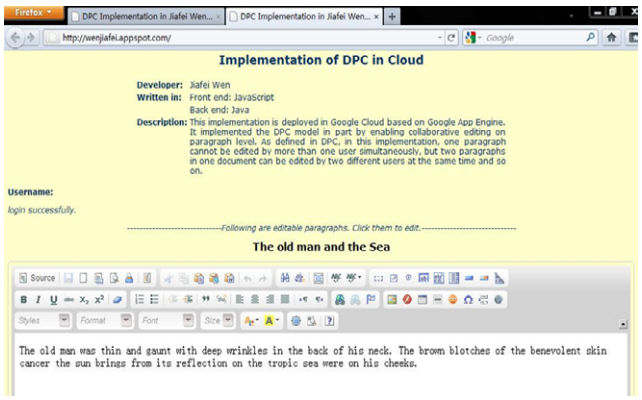

Figure 12: Collaborative Editing through Different Browsers.

DPC model provides a proper granularity of collaborative editing by eliminating editing confusion and disorder among users.

References

1. Adler A, Nash JC, Sylvie N (2006) Evaluating and implementing a collaborative office document system. Interacting with Computers 4: 1-18.
2. Buyya R, Yeo CS, Venugopal S, Broberg J, Brabdic I (2009) Cloud Computing and Emerging IT Platforms: Vision, Hype, and Readily for Delivering Computing as the 5th Utility. Future Generation Computer Systems 6: 599-616.
3. Kim E, Severinson Eklundh K (1998) How Academics Co-ordinate their Documentation Work and Communicate about Reviewing in Collaborative Writing. Technical Report TRITA-NA-P9815, NADA.
4. Newman J, Newman R (1992) Three modes of collaborative authoring in Computers and Writing: State of the Art (P.O. Holt and N. Williams, Eds.). Oxford: Intellect Books 20-28.
5. Noel S, Robert JM (2003) How the Web is used support collaborative writing. Behaviour & Information Technology 22: 245-262.
6. Noel S, Robert JM (2004) Empirical Study on Collaborative Writing: What do co-authors do, use, and like? Computer Supported Cooperative Work. The Journal of Collaborative Computing 13: 63-89.
7. Petride S, Tarachandani A, Agarwal N, Idicula S (2011) Managing and Processing Office Documents in Oracle XML Database. In Proc. Of the 3rd International Conference on Advances, Knowledge, and Data Applications 89-95.
8. Posner IR, Baecker RM (1992) How people write together, in Readings in Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration (R.M. Baecker, Ed.) 239-250.
9. Rizzi CB, Alonso CMMC , Hassan EB , Tarouco LMR, De Seixas LMJ (2000) EquiText: a helping tool in the elaboration of collaborative texts. Proc. of 2000 Society for Information Technology and Teacher Education International Conference 2314-2319.
10. Sun C, Ellis C (1998) Operational transformation in real-time group editors: issues, algorithms, and achievements. Proc. of the 1998 ACM conference on Computer supported cooperative work. 59-68
11. Tang Y, Yan C, Suen CY (1994) Document Processing for Automatic Knowledge Acquisition. Proc. of IEEE Transactions on Knowledge and Data Engineering 6: 3-21.
12. Wen J, Lam S, Wu X (2011) A model for Office document processing and collaboration in the cloud. Proc. of the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications.
13. Zhang L, Zhou Q (2011) CCOA: Cloud Computing Open Architecture. Proc. of IEEE International Conference on Web Services.
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) (2007) 2.0. W3C Recommendation.
17. http://www.w3.org/TR/xpath20/
18. http://jquery.com/
19. http://ckeditor.com/what-is-ckeditor
20. http://ckeditor.com/license
21. http://www.w3.org/TR/xslt
22. http://en.wikipedia.org/wiki/Google_App_Engine