REAL TIME COLLABORATIVE EDITING IN MOBILE COMPUTING ENVIRONMENTS

Cristina-Loredana DUȚĂ[1], Laura GHEORGHE[2], Nicolae ȚĂPUȘ[3]

Nowadays, due to the rapidly development of technology, the mobile computing environment is in continuous transformation. The purpose of this paper is to determine how mobile devices can be used for the development of collaborative systems. The field of mobile collaborative work is in progress (is a part of the Computer Supported Collaborative Work - CSCW) and will provide in the future a tool for co-workers to collaborate and work on the same shared workspace in real-time to provide data such as text, images and diagrams regardless of their geographical location. This paper describes several algorithms used to implement group editors such as: causal order algorithm based on vector clocks, total order algorithm based on a centralized component, total order algorithm based on a token, three-phase distributed algorithm, dOPT, Jupiter and Paxos. A comparative performance evaluation is made to determine which of these algorithms is the most suited for mobile environments where the network is an important and expensive resource.

Keywords: distributed systems, concurrency control, collaborative editing, mobile computing, operational transformation, vector clocks

1. Introduction

The mobile computing environment is in continuous transformation ranging from laptops to mobile phones, from digital cameras and players to portable devices. The new devices need an environment with different characteristics than the older mobile management systems. To achieve this, users must be able to carry their data with them all the time, regardless of their geographical position and also to be able to access and modify it whenever they need. The purpose is to determine how mobile devices can be used for the development of collaborative systems. When referring to real-time collaborative editors there are two types of architectures which can be used: a centralized or a replicated architecture.

[1] PhD student, Department of Computer Science and Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: cristina.duta.mapn@outlook.com
[2] Assistant Professor, Department of Computer Science and Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: laura.gheorghe@cs.pub.ro
[3] Professor, Department of Computer Science and Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: nicolae.tapus@cs.pub.ro

In a **centralized architecture**, a central server holds the shared document and resolves problems such as ordering of updates, conflicts and document consistency. Every user will send his action to the server and the server will apply it to the document.

At the other end is the **replicated** or **decentralized architecture** where every site owns a copy of the shared document on which they work independently. Every user executes his operations on the local replica immediately without delays and then the operations are transmitted to the other sites. Because a user has to be able to work on the local replica even if he is disconnected for a period of time, this last solution is preferred in mobile networks which have limited resource reliability and availability.

In this paper, we present seven types of protocols for maintaining consistency in real-time collaborative text editors and we perform a comparative evaluation in order to determine which is the most appropriate for mobile environments. The paper is organized as follows. Related work is presented in Section 2. Section 3 gives an overview of the protocols chosen to maintain consistency. Section 4 offers details about our implementation. In Section 5 shows the experimental results obtained using a testing simulator to generate random traffic. Section 6 draws the conclusions for the protocol comparison and presents future work.

2. Related Work

Even though people have been using editors for a while now, not many of them took into consideration the wide variety of interesting research issues regarding an editor used in real collaborative context. For these applications, one of the main challenges remains consistency maintenance, which is so complex that it has a growing research potential. To offer a better perspective and understanding of our work, this section presents the results obtained by other researchers.

The first presentation of collaborative real-time editor was made by Douglas Engelbart in 1968, in "The Mother of All Demos" [1], but the actual implementations of such an editor appeared many years after. In 1991, Mac OS revealed Instant Update [2] and later, a version for Microsoft Windows was released as well, which could provide real-time collaboration between the two previously mentioned operating systems. Their product was based on a centralized server which coordinates the documents updated by several clients in real time. When Web 2.0 appeared, a product called *Writely* [3] had an explosive growth and was acquisitioned by Google in 2006 (represents today's Google Docs). Other two products, such as *Synchroedi*t [4] and *MobWrite* [5] attempted to solve the

problem of real-time browser-based collaborative editing, but they were unable to achieve true real-time performance, especially for large scale architectures.

The difficulty of a real-time collaborative editing system relies in the size of the communication latency. The problem that appears is that clients must see their edits inserted instantly into the document, but in this case due to communication latency, their modifications must be inserted into different versions of the document. Thus to develop a real-time collaborative text editing for mobile computing requires methods for content adaption, reduced usage of network bandwidth and conflict resolution mechanisms. We will mention further on some of the efforts that address the need for concurrency control in collaborative applications.

The most frequently used technique to ensure consistency is the operational transformation method. The main advantage for mobile devices is that users can work on local data replicas, even if they are disconnected. In [6], the authors present a review of several operational transformation techniques (dOPT, GOT, aDOPTed, GOTO) in order to identify the issues, algorithms, achievements and what challenges remain still unresolved. After taking all these elements into account, they propose a new optimized generic operational transformation algorithm. A comparison between different algorithms is also made in [7]. dOPT, adOPTed, GOT, GOTO, SOCT2, SOCT3, SOCT4, SDT, ABT and ABTS are compared based on criteria such as: correctness, property of operations of remote sites, storage, time complexity, transformation function, space complexity. Even though there are many similarities between these algorithms, the authors try to emphasize the advantages and drawbacks of each of them, in order to identify major issues for further research.

In [8] the authors describe a new transformation-based merging algorithm which supports mobile collaboration. They compare it with other optimistic consistency control methods and conclude that their algorithm improve the existing time complexity and ensure updates without loss.

Compared to our work, there are some differences, because they analyze only operational transformation methods, while we analyze besides the dOPT algorithm, different types of consistency maintenance methods and we compare their suitability for mobile environments.

Similar approach with how we defined and implemented the operations to evaluate the methods used for collaborative editing is presented in [9]. They focused their research in studying the existence of transformation functions that satisfy two properties which are necessary and sufficient to ensure convergence. They applied these transformation functions to shared strings which can be modified (by insert and delete operations).

Even though many protocols which solve agreement problems have been published, little has been done to analyze their performance. Our paper focuses on

analyzing the performance of different types of algorithms used to implement group editors in order to determine which is best suited for mobile environments.

In [10], the authors focus on comparing the performance of consensus algorithms such as Paxos and Chandra-Toueg. For their experiment they varied the number of processes involved in the execution and determined the latency based on different classes of runs (with failures or with no failures). Even though for most of the cases the two algorithms had the same performance, Paxos is more efficient when the process that handles the first round of the protocol crashes. Some strategies to support collaborative editing are described in Section 3.

3. Algorithms

In a distributed system there are different types of processes that communicate by sending messages to each other. In this situation, an order for the generated events must be imposed. This means that the distributed algorithm has to take into account all the rules defined in [11]. One solution is a causal order algorithm based on logical clocks.

3.1. Causal order algorithm based on vector clocks

The first protocol presented is based on vector clocks [12] and is an algorithm used to ensure partial ordering of events in a distributed system and to detect causality violations. When using vector clocks we must ensure a *delivery protocol* so that every site has the capability to delay received messages if it is necessary and to deliver them in a consistent order. In our implementation we used vector timestamps. The delayed messages are stored in a queue and this is sorted by vector time and the concurrent messages are ordered based on the time of receiving them.

Even though *causal order* is most commonly used, for situations such as updating replicated data in distributed systems *total order* is a better option, because it requires all the messages that were sent to arrive at the receivers in the same order. For the total order protocol we have implemented three algorithms as described further on.

3.2. Total order algorithm based on centralized component

This solution is very useful for systems with FIFO channels. Every site that wants to transmit a message sends it only to the centralized component, which simply broadcasts all the messages it receives to all the other sites included in the system. Because the server receives the messages in a certain order, the order is given by the server and that is how the messages are transmitted further to the other sites. The main disadvantage of the solution based on a centralized component is that it has a single point of failure and congestion, which at some moment will create problems.

3.3. Total order algorithm based on a token

These token based algorithms are very efficient in terms of throughput (the number of messages that can be delivered per time unit). This is justified because they are able to reduce the network contention by using the token to eliminate the problem of ACK transmissions and to ensure flow control.

The protocol works as follows. When a site has generated a message, it cannot send it immediately. It will put its messages in a queue and will start to deliver them when it gets the token, for as long as it has the token. If messages still remain in the queue, they will be sent the next time the site gets the token. In this way a total order is ensured similar to a serialized form. We have implemented a total ordering protocol based on a dynamic token-passing scheme which determines the next token holder dynamically, not in predetermined order.

3.4. Three phase distributed algorithm

In three phase distributed algorithm, the site is both sender and receiver and each site has a queue where it stores the messages received. When a site is the sender it follows the next three phases. In the first phase, the site broadcasts its message M with a locally unique tag and the local timestamp to all the other sites in the system. In the next phase, the sender waits for replies from all the receiver sites. Then, it will determine the maximum of the *proposed timestamps* it received for message M and declares the value chosen as the final timestamp. In the last phase, the sender will broadcast the final value to all the sites in the system, including himself.

If the site is a receiver it will execute the following steps. First, it will receive the message M with a proposed timestamp. In the second step, the receiver site will send the revised value and a tag back to the sender of the message M. In the third step, it waits to receive the final timestamp. Based on this value, the receiver replaces the revised timestamp with the new received value and reorders the queue. If the message M represents the head of the queue, it will be delivered; otherwise it will wait until it becomes the head of the queue and then delivers it. The main advantage is that the messages can be delivered by all the sites in the same total order.

3.5. dOPT algorithm

The dOPT algorithm [13] uses a replicated architecture were the documents are reproduced at each participating site and the copies are initially identical. The algorithm works as follows. When a site generates an operation, it executes it locally immediately, generates a priority for the operation and then sends these pieces of information to all the other sites in the system. When a site receives an operation, the information from the message received is examined. If there have been operations executed by the site which sent the message, the

operations will be queued, otherwise the operations will be executed. After all of these, the algorithm updates its copy of the document to reflect the transformed update.

dOPT offers several advantages for collaborative text editors such as ensuring consistency among distributed replicated documents without the need of serialization or even serializability among the updates at different sites. Although dOPT is a simple algorithm that satisfies many of the correctness properties, Gordon Cormack [14] found a case where dOPT could not always ensure convergence when remote concurrent requests with similar operations are transmitted from two different sites. After further research a solution to this problem was found: to use different data structures for time stamping and conflict handling (for example, the Jupiter algorithm) or to transform the log entries every time they are used to transform an update.

3.6. Jupiter algorithm

Jupiter [15] is a multimedia, multi-user virtual world which offers support for long-term remote collaboration. Jupiter is different from other groupware systems because it does not use the synchronization protocol directly between the sites, but synchronizes each client with the server and then the server will make all the changes and send the changes made by one site to the other sites in the system.

In the Jupiter protocol, each message will be labeled with the state of the sender, before the generation of the message. This component is used for conflict detection and then *xform* function is used to solve these conflicts. The algorithm can ensure that regardless of the divergence between client and server in the state space, they will have identical values when they reach the same state.

The Jupiter protocol fixes the problem with the dOPT algorithm. In case of dOPT, when a message from a site diverges more than one step in the state space, the algorithm will not transform the saved messages when it processes the new incoming messages. Jupiter protocol treats the N-way consistency in a different manner, so the problem previously mentioned never appears for it.

3.7. Paxos algorithm

Paxos [16] is a protocol for distributed systems where sites in the same group can use to agree on a value proposed by a member of the group. At the end, the algorithm reaches consensus regardless the unreliability of the network and the simultaneously multiple attempt proposals of different values.

A site can be any of the following components: *proposer* (proposes a value), *acceptor* (accepts or rejects a proposed value) and *learner* (is informed about the chosen value). The steps in Paxos protocol are described below.

The *proposer* tries to confirm a proposed decision value (which is selected from an arbitrary input group) by collecting approvals from a majority of *acceptors*, while the *learners* have the role to observe how this approval is done. By ensuring that only one of the proposals can receive the votes of a majority of *acceptors*, the agreement is enforced. Moreover, the validity is ensured by allowing only input values to be proposed. Any *proposer* can decide to restart the protocol by issuing a new proposal. In this case, the algorithm ensures a procedure to release the *acceptors* of their old votes.

4. Application Implementation

The sites in a groupware system communicate through TCP/IP connections. Our application provides unconstrained group editing of documents between users based on different protocols: casual order algorithm, total order algorithm with a centralized component, total order algorithm using tokens, dOPT, three-phase distributed algorithm, Jupiter and Paxos. We used *NetBeans IDE 7.3* and *Java* programming language for developing the application interface and the protocols which ensure consistency between the sites in a collaborative text editor. Fig. 1 shows how the window for editable documents in our application appears to the user.

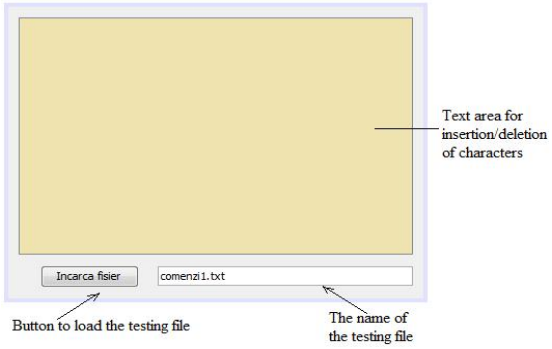

Fig. 1. The window of the application

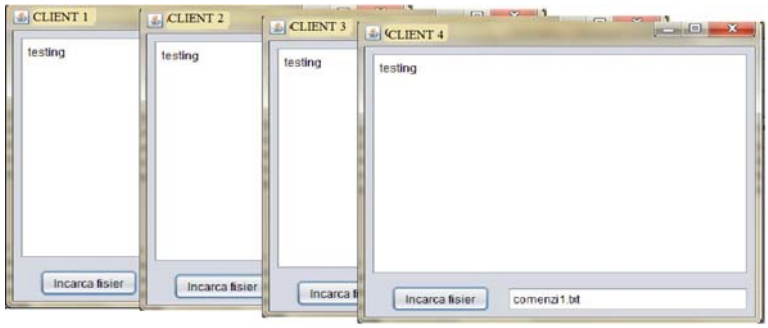

Fig. 2. Four clients connected at the same time

The window in Fig. 1, consists of a *TextBox* which displays the state of the document and allows users to perform different operations, a pushbutton named *"Load File"* which loads the testing file for the application and a *TextPanel* which is displaying the name of the testing file used by the application. The application, as it can be seen in Fig. 2, allows two or more users to remotely edit a document simultaneously. The editing is completely unconstrained and users can insert and delete characters at any location.

To manipulate a document, the user can perform (automatically, by loading the file *commands.txt*) two operations: ***INSERT (char,pos)*** (inserts a new character to the specified position in the document. If the position has not been reached yet, the character is appended at the end of the written text) and ***DELETE (pos)*** (deletes a character situated at the specified position in the document).

Fig. 3 shows how the windows of clients appear when a total order based on a token protocol is used. For optimistic algorithms such as dOPT and Jupiter the changes are reflected immediately in the user's window, before they are processed by the other sites.

Fig. 3. Window for token based protocol

For Paxos algorithm, several messages are displayed. If a site is *proposer*, the message *"This site is a proposer"* appears in the console together with the proposed value, the number of acceptors that are in the system at the current moment and if the value has been accepted or not. If a site is an *acceptor*, the message *"This site is an acceptor"* appears in the console, together with the value received from the proposer. If the value is the chosen value, the message *"Propose OK"* is displayed and the acceptor will send this value to the distinguished learner. If the site is a *distinguished learner* it receives the value chosen from all the acceptors and then sends it to all the other learners. The message *"This site is a distinguished learner"* or *"This site is a learner"* appears in the console.

5. Experimental evaluation

For our experiment, we used a system with Intel Core Dual CPU 2 GHz, 2 GB of RAM, Windows 7 32-bits and integrated video card. The purpose of this paper is to determine which of the protocols for maintaining consistency is more suitable for mobile environments, where the network is an important and expensive resource. We took into consideration the total number of bytes sent through the network for each method and the number of users involved (which varies from 2 to 6) and we used a testing module to generate random traffic between the users. This module automatically performs different operations (insert, delete characters) for each client based on predefined files which contain these operations. The results were plotted using Java's Swing widget for NetBeans IDE 7.3. The experiments that were performed are presented in detail in the following sections.

5.1. Experimental results for files of 10, 50 or 100 characters

In Fig. 4 we have a graphical representation of the experimental results for all the algorithms previously mentioned, obtained for small files of 10 characters and a various number of clients (from 2 to 6 clients).

It can be observed that the number of bytes sent through the network by each algorithm is different according to the number of clients. For two clients, the causal order algorithm based on vector clocks offers the best results (all 10 bytes are sent). Also, the number of bytes sent by the total order algorithm based on a centralized component is high (10 bytes for two clients and 9 bytes for six clients). It is a fast algorithm which ensures a maximum number of bytes sent through the network at once.

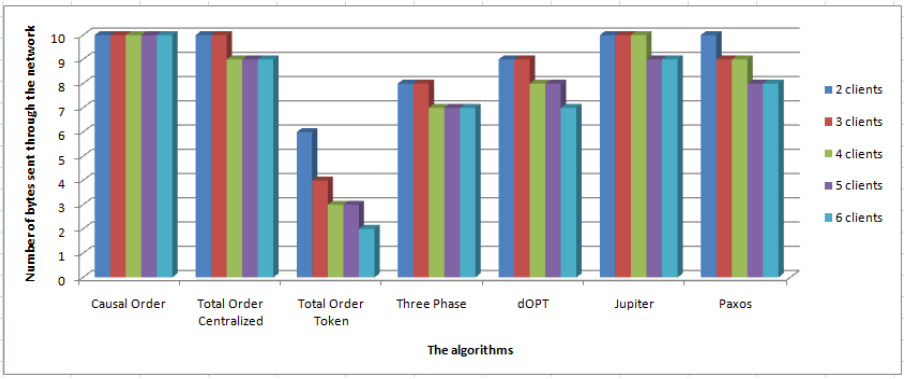

Fig. 4. Number of bytes sent for files of 10 characters

In Fig. 5 and Fig. 6 are shown the number of bytes sent by the algorithms through the network, when using small files of 50 and 100 characters respectively, and different number of clients.

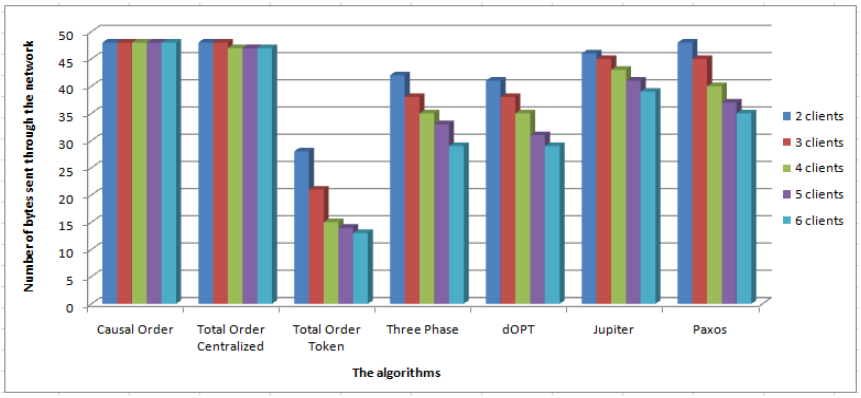

Fig. 5. Number of bytes sent for files of 50 characters

For the first two algorithms it can be seen that the number of bytes sent doesn't decrease very much (96 bytes of 100 are being transmitted, which is very good). The causal order algorithm offers a very satisfying result, when we deal with FIFO channels.

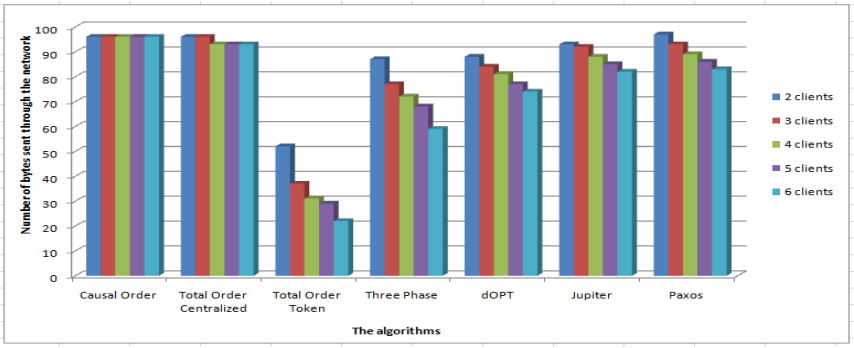

Fig. 6. Number of bytes sent for files of 100 characters

However in real life we have no FIFO channels and two messages generated from the same site can arrive at the other site in reverse order. In this case, the second message (which arrived first) cannot be executed before the first message because it depends on it.Comparing three-phase distributed algorithm with the previous algorithms it can be observed that is more complex and has a lower number of bytes sent through the network (87 bytes sent for two clients and 59 bytes for six clients). This happens because it needs 3N messages to send a

message to N sites and also includes a delay of 3 message hops (corresponding to the three phases).

For the total order algorithm based on a token, the number of bytes sent through the network is inverse proportional with the number of clients: as the number of clients participating in the application increases, the number of bytes sent by them is decreasing. This situation is justified because, the token is dynamically passed to clients and when the number of users increases, the chance to get the token becomes smaller.

5.2. Experimental results for files of 250 and 500 characters

Fig. 7 and Fig. 8 show the number of bytes sent through the network by all algorithms when using a different number of clients and files of 250 and 500 characters respectively.

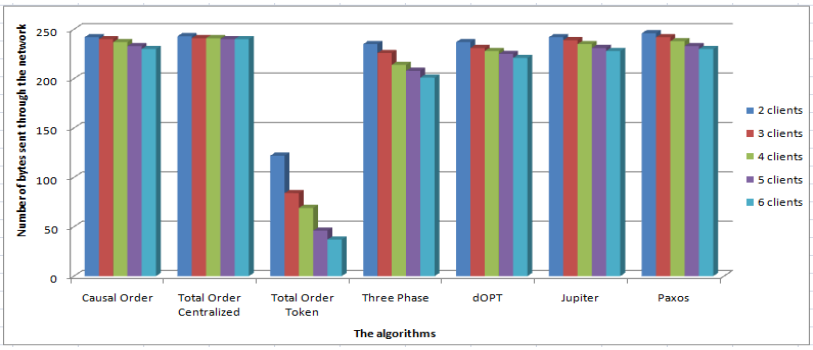

Fig. 7. Number of bytes sent for files of 250 characters

The results of the total order algorithm based on a centralized component are very good, but the main disadvantage of this protocol is that is a single point of failure and if the server crashes, there will be no communication. Because in a mobile environment the connectivity may be intermittent, the transmission may be interrupted and the central server may be unreachable, this algorithm doesn't seem the best solution for collaborative editing in mobile environment.

It can be seen that the total order algorithm based on a token has the worst results (only 72 bytes out of 500 are sent for six clients). This is not a solution to handle consistency in mobile environments because this requires a large usage of the network bandwidth, which is limited for this type of devices.

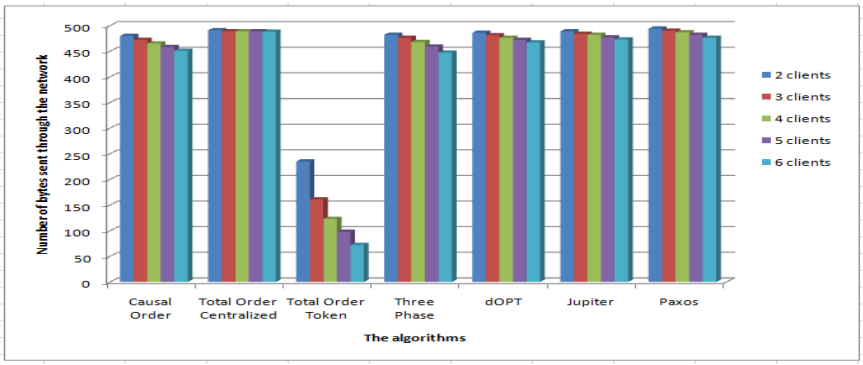

Fig. 8. Number of bytes for files of 500 characters

The three-phase distributed algorithm has good results (447 bytes out of 500 for six clients), but if we take into consideration the complexity of the algorithm and the time needed to send messages to all the other sites, we can affirm that it may be a solution for mobile environment, but for sure not the best.

For dOPT algorithm, the differences in the number of bytes sent through the network when having two clients (486 bytes sent) to six clients (467 bytes sent) are small. Unfortunately, even if this seems the best solution for a mobile environment because it allows the users to work on local data replicas even in disconnected mode, a scenario was found where dOPT could not always ensure convergence, when remote concurrent requests with similar operations were transmitted from two different sites.

For Jupiter algorithm, results are better than for dOPT algorithm, 489 bytes sent for two clients and 473 bytes for six clients. Jupiter is appropriate to use for mobile environments, but the problem relies in the existence of the server, which is a single point of failure and which will interrupt communication between sites when it crashes.

For Paxos algorithm with two clients, 494 bytes out of 500 are transmitted and for six clients 476 bytes are sent through the network. Even though it has great results, Paxos is not suitable for collaborative editing in mobile environments, because if the network is down, there is no chance for all acceptors to reach an agreement and to choose a proposed value.

6. Conclusion

The aim of this paper is to compare several algorithms which maintain consistency in collaborative editing applications and decide which can be used in mobile environments. According to the simulation results we can conclude that algorithms which include operational transformation (dOPT and Jupiter) are the most suited to support collaboration using mobile devices because it allows users

to work on local data replicas even in a disconnected mode and synchronize with each other when reconnected. Our future work will involve evaluating other operational transformation algorithms (such as adOPTed, GOTO, ABT) and discover which is the best for collaborative editing in mobile environment.

Acknowledgement

This work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/134398.

R E F E R E N C E S

[1]. *D. Engelbart*, 1968 demo, http://vimeo.com/1408300 (Accessed: December 2014).
[2]. *PayPal – Instant Update API*, 2009, http://cms.paypal.com/cms_content/US/en_US/files/developer/PP_Callback_InstantUpdateAPI_Guide.pdf (Accessed: December 2014).
[3]. *Writely Word Processor,* 2005, http://disedlibrarian.edublogs.org/2005/12/08/writely-word-processor/ (Accessed: December 2014).
[4]. *Synchronous Editing for the Web,* 2006, http://www.synchroedit.com/ (Accessed: December 2014).
[5]. *MobWrite*, 2010, http://www.mobwrite.net/static/about.html (Accessed: December 2014).
[6]. *C. Sun and C. Elli*, "Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achivements", in Proceedings of 1998 ACM Conference on Computer Supported Cooperative Work, 1998, pp. 59-68.
[7]. *S. Kumawat and A. Khuntela*, "A Survery on Operational Transformation Algorithms: Challenges, Issues and Achievements", in International Journal of Computer Applications, 2010, Vol. 3, pp. 30-38.
[8]. *B. Shao, D. Li and N. Gu*, "A Fast Operational Transformation Algorithm for Mobile and Asynchronous Collaboration", in Parallel and Distributed Systems, IEEE Transactions, 2010, Vol. 21, pp. 1707-1720.
[9]. *A. Randolph, H. Boucheneb, A. Imine and A. Quintero*, "On Consistency Operational Transformation Approach", in Infinity'12 EPTCS 107, 2013, pp. 45-59.
[10]. *N. Hayashibara, P. Urban and A. Schiper*, "Performance Comparison Between the Paxos and Chandra-Toueg Consensus Algorithms", in Proceedings of International Arab Conference on Information Technology, 2002, pp. 22-34.
[11]. *R. Hirschfelder and J. Hirschfelder*, "Introduction to Discrete Mathematics", Wadsworth Pub Co, 1990.
[12]. *L. Lamport*, "Time, Clocks and the Ordering of Events in a Distributed System", in Communications of the ACM, No. 7, 1978, pp. 558-565.
[13]. *C. A. Ellis and S. J. Gibbs,* "Concurrency control in groupware systems", in ACM SIGMOD Record 18, 1989, pp. 399-407.
[14]. *G. V. Cormack*, "A Counterexample to the Distributed Operational Transform and a Corrected Algorithm for Point-to-Point Communication", in Research Report CS-950-08, University of Waterloo, 1995.

[15]. *D. A. Nichols, P. Curtis, M. Dixon and J. Lamping*, "High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System", in Proceedings of the ACM User Interface Software and Technology Symposium, 1995, pp. 234-244.
[16]. *L. Lamport*, "Paxos Made Simple", in ACM SIGACT News, No. 4, 2001, pp. -58.