Concurrency Control for Mobile Collaborative Applications in Cloud Environments

Moulay Driss Mechaoui and Abdessamad Imine

Abstract As the world is progressing quickly towards more connected mobile devices, the use of mobile collaborative applications is gaining an increasing popularity. For instance, real-time data streams and web applications (such as social networking and ad-hoc collaboration) are seamlessly incorporated in mobile applications. Despite this powerful evolution, the resource limitation (energy consumption and unstable connectivity) remains a serious problem against a safe concurrency control for an efficient and continuous use of mobile collaboration. In this chapter, we describe the data consistency issues when mobile applications support collaboration through the cloud. Based on human factors (such as high interactivity and data consistency), we present two concurrency control techniques for offloading and ensuring data synchronization among mobile devices and the cloud. The first technique relies on a client-server style to ensure safe coordination, while the second one supports a peer-to-peer mechanism to achieve a decentralized data synchronization.

1 Introduction

The spectacular development of mobile devices (smartphones, tablets, PDA) and the rapid progression of mobile communications in these last few years have offered a new environment of development for mobile applications. These mobile devices have changed the way we interact with our social environment and become the devices of choice to collaborate with family members, friends and business colleagues and/or customers. However, deploying ad-hoc collaboration around mobile applications requires increasing amounts of computation, data storage and network communica-

M.D. Mechaoui (✉)
University of Sciences and Technology Oran 'Mohamed Boudiaf' USTO-MB, Oran, Algeria
e-mail: moulaydriss.mechaoui@univ-usto.dz

A. Imine
Université de Lorraine and INRIA-LORIA Grand Est, Nancy, France
e-mail: abdessamad.imine@loria.fr

C.X. Mavromoustakis et al. (eds.), *Advances in Mobile Cloud Computing and Big Data in the 5G Era*, Studies in Big Data 22,
DOI 10.1007/978-3-319-45145-9_11

tions. Moreover, preserving the consistency of the manipulated shared data (such as the shared document in mobile collaborative editor) under constraints of the mobile applications, namely the freshness and the energy consumption, remains still a serious problem.

In this case, resorting to cloud computing becomes a necessity. Cloud computing is a multi-purpose paradigm that aggregates several technologies such as virtualization, peer-to-peer networks and autonomic computing. It is an emerged model based on virtualization for efficient and flexible use of hardware assets and software services over a network. Virtualization extends the mobile device resources by offloading execution from the mobile to the cloud where a clone (or virtual machine) of the mobile is running. It provides a seamless and rich functionality to mobile applications regardless of the resource limitations of mobile devices. Cloud computing allows users to build virtual networks "à la peer-to-peer" where a mobile device may be continuously connected to other mobiles to achieve a common task.

In this chapter, we provide a global view of mobile collaborative applications in the cloud, while highlighting the specific issues of data consistency in mobile cloud computing. We present the principle and drawbacks of two concurrency control techniques (centralized and decentralized) for offloading and preserving data consistency between mobile devices and the cloud. More precisely, we describe the components of two existing collaborative editing protocols, CloneDoc [1] for the centralized control concurrency and OptiCloud [2, 3] for the distributed one.

The remainder of this chapter is organized as follows: Data consistency issues related to mobile collaboration through the cloud are given in Sect. 2. Section 3 presents a concurrency control scheme supporting the client-server style for consistency maintenance of the shared data. In Sect. 4, we describe another concurrency control scheme based on a pure peer-to-peer model. We discuss the related work in Sect. 5 and conclude in Sect. 6.

2 Data Consistency Issues

In this section, we present the collaborative model for manipulating shared data regardless of spatial and temporal constraints using mobile devices in the cloud environment, and we illustrate this model by two use cases. Finally, we highlight data inconsistency problems that mobile users are likely to face due to mobile-to-clone and clone-to-clone interactions.

2.1 Collaboration Model

The aim of the collaboration in cloud environments is to allow many geographically dispersed users to manipulate the shared data at anytime and anywhere. This collaboration model involves a set of mobile devices and a set of clones (or virtual

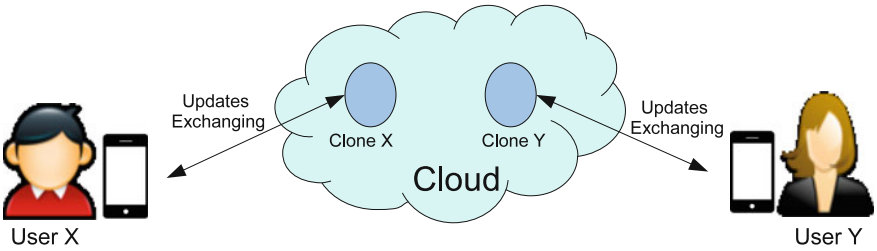


Fig. 1 Collaboration model

machines) in such a way each mobile user owns her/his clone in the cloud as illustrated in Fig. 1. The clone is characterized by machine features (e.g. CPU frequency, memory size) and a virtual image to be set up (e.g. softwares like the operating system and synchronization protocols).

To achieve continuous collaboration with high data availability, each user owns two copies of the shared data where the first one is stored in the mobile device whereas the second one is on its clone (at the cloud level). The collaboration between users is performed as follows: each user modifies the mobile copy and then sends local modifications to her/his clone in order to update the second copy and propagate these local modifications to other mobile users by means of their clones.

Based on human factors, a mobile collaborative application is characterized by the following requirements [2, 3]:

1. *High local responsiveness*: the application has to be as responsive as if it is based on single-user;
2. *High concurrency*: users must be able to concurrently and freely modify any part of the shared data at anytime and anywhere;
3. *Consistency*: all concurrent updates must be synchronized in such a way users must eventually be able to see an identical view of all copies;
4. *Scalability*: a group must be dynamic in the sense that users may join or leave the group at any time;
5. *Failure recovery*: users have to recover easily all shared documents when technical hitch (e.g. crash, theft or loss of mobile device) happens, and continue seamlessly the collaboration.

In this chapter, we focus only on features 2 and 3 to present how existing systems achieve these features.

In the following, we present two use cases that illustrate how this collaboration model can be deployed over mobile cloud network to overcome some problems:

Assisting tourists. Suppose that a group of tourists want to make a tour in London city. They are already provided with a mobile application based on the proposed collaboration model helping them to visit the city using a map. One member of the group creates a collaborative group in the cloud and downloads the map with relevant information about the city. The other members join the created group and share the

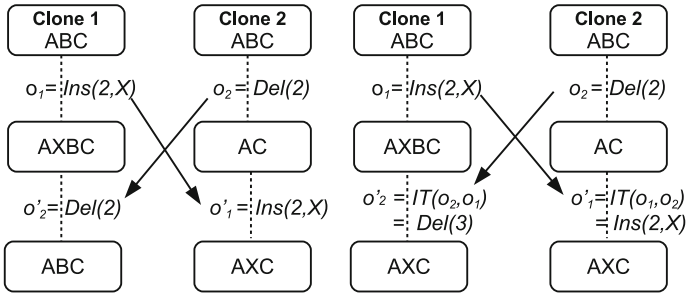
London map. When visiting museums, restaurants or historic places, tourists can share, in real-time, their opinions by writing their comments in the local map. This allows enriching and updating the map content. For instance, during the journey one of them realizes that, instead of the art gallery appearing on his map, there is a pharmacy. Hence, she/he corrects/updates the description of this localization in her/his map, and then she/he sends it to her/his clone in the cloud in order to be broadcast to other group members. At the meantime, one tourist was disconnected when sending the update information. In this case, her/his clone will notify the update information after her/his re-connection to the group as if she/he did not quit it.

Social networking. With the widespread use of online social networks (e.g. Facebook, MySpace, and Twitter), which have become a lifestyle in our society, allowing users to keep in touch with families and friends and also extending for business purposes such as searching for new career opportunities. Recently, these social networks are increasingly going mobile, and propose a new trend of social networks called Mobile Social Networks (MSN) [4–6], that has emerged and attracted considerable attention from the academic and industrial communities. A user can share and synchronize social data (such as a list of friends, comments, a set of song lyrics, etc.) across a group of friends. However, MSN should address the constraints of mobile devices, i.e., limited energy, low memory capabilities, limited processing power, scalability, and heterogeneity [7]. To satisfy the constraints of mobile devices in MSN, a cloud network can be used where each mobile device creates its own clone. The clone stays connected and reachable for the other clones in the cloud whether its mobile device is connected or not. The processing, storage and dissemination of data are delegated to the clone. The clone can also send possible updates/notifications (e.g. recommendations, nearby friends, prizes, etc.) back to the other clones or to its mobile device either when requested or as a response to the events created by other mobile devices. This cloning-based model does not require the mobile device to be online all the time. Therefore, it retains the power consumption of the mobile device.

However, this collaborative model is not free from problems. The main challenge is: how to maintain consistency and properly resolve conflicts throughout the shared data, while several users are simultaneously updating the same data? Data inconsistency may appear in situations related to clone-to-clone and mobile-to-clone interactions.

2.2 Interaction Between Clones

The inconsistency problem occurs when two or several clones produce simultaneous updates. To illustrate this problem, consider the following example of collaborative editor where two clones, CLONE 1 and CLONE 2, contain the same document "ABC". CLONE 1 executes editing operation $o_1 = Ins(2, X)$ to insert the character 'X' at position 2 and ends up with "AXBC". Concurrently, CLONE 2 performs editing operation $o_2 = Del(2)$ to remove the character 'B' at position 2 and obtains the



(a) Wrong Integration *(b) Correct Integration*

Fig. 2 Clone to Clone synchronization scenario

state "AC". After exchanging operations among clones, the CLONE 1's document becomes "ABC" but the CLONE 2's document has a different state "AXC" as shown in Fig. 2a.

Several cloud-based collaborative editors such as Google Docs,[1] Cloud9,[2] Zoho Suite[3] use Operational Transformation (OT) approach [8, 9] that provides a general model for synchronizing the shared data, while allowing each user to apply local updates optimistically. OT is considered as the efficient and safe method for consistency maintenance in the literature of collaborative editors. It consists to transform the parameters of an operation to include the effects of previously concurrent operations so that the transformed operation can lead to consistent document. In the OT approach, each site is equipped by two main components [8, 10]: the integration component and the transformation component. The integration component determines how an operation is transformed against a given operation sequence (e.g., the log buffer). It is also responsible for receiving, broadcasting and executing operations. It is rather independent of the type of the shared data. The transformation component is a set of *transformation functions* which is responsible for merging two concurrent operations defined on the same state. Every transformation function is specific to the semantics of a given shared data. The most known OT-based theoretical framework is established by Ressel et al. [10]. They define two consistency criteria:

- *Causality*: If one operation O_1 causally precedes another operation O_2, then O_1 must be executed before O_2 at all sites.
- *Convergence*: When all sites have performed the same set of operations, the copies of the shared data must be identical.

[1] https://www.google.fr/intl/fr/docs/about/.

[2] https://c9.io/.

[3] https://www.zoho.com/.

Thus, at CLONE 1, operation o_2 needs to include the effect of o_1 using a transformation function IT: $o_2' = IT(o_2, o_1) = Del(3)$. As for CLONE 2, operation o_1 is left unchanged. Accordingly, both of them get the same state "AXC" as shown in Fig. 2b.

2.3 Interaction Between Mobile and Its Clone

Since the mobile and its clone are two entities that are physically and geographically separated. Each entity has its own local copy of the shared data to be synchronized. Therefore, a delay of applying the same operations on both sides with the same copy is possible. This may lead to document inconsistency.

Consider the example of collaborative editing illustrated in Fig. 3. Given a mobile M and its clone C that have the same initial document "ABC". Mobile M performs a local update operation $o_1 = Ins(2, X)$ to insert the character 'X' at position 2 and results in state "AXBC". Simultaneously, clone C executes two operations o_2 and o_3 coming from other clones: operation $o_2 = Ins(2, X)$ adds the character 'X' at position 2 and gives the state "AXBC"; operation $o_3 = Del(3)$ removes the character "B" at position 3 and obtains the state "AXC".

When Mobile M and its Clone C decide to synchronize, they commit their operations that have been applied locally. After exchanging operations, clone C performs operation o_1 and its document becomes "AXXC". At the meanwhile, Mobile M executes operations o_2 and o_3 to result in the state "AXBC". As illustrated in Fig. 3, the mobile and its clone have different states.

It is clear that preserving data consistency between the mobile and its clone requires some additional treatments that have to be performed by either the mobile

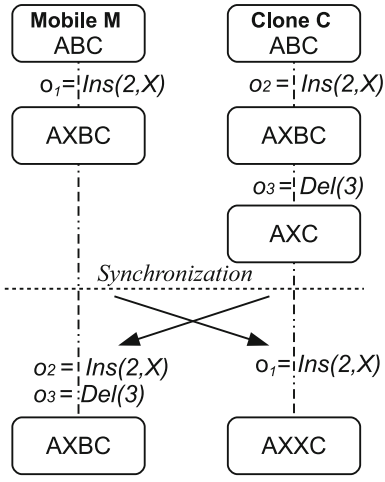


Fig. 3 Mobile to clone synchronization scenario

device, its clone or both. Note that the more computing tasks are executed on mobile device the more battery life is reduced.

To deal with data consistency problems caused by clone-to-clone and mobile-to-clone interactions, two kinds (centralized and decentralized) of OT-based concurrency control protocols will be presented in the following sections.

3 Centralized Concurrency Control

In this section, we give a general presentation of concurrency control protocols designed in the client-server style and using OT approach to coordinate all concurrent updates.

3.1 Principle

To maintain consistency, many centralized concurrency control protocols (such as CloneDoc [1], SPORC [11] and Google Docs[4]) are based on a single server (or super clone), that is the backbone of the collaboration with the following features:

- It relies on OT technique to enforce continuous and global order on all updates to avoid the divergence of user's document view from the server.
- It enables users to join or/and leave any collaborative group;
- It ensures the availability of shared documents for all mobile users.
- It manages the synchronization and propagation of updates between clones.

Two layers are used to maintain the data consistency: the first layer ensures synchronization between clones and the second one consists in synchronizing the mobile with its clone.

Clone-to-Clone synchronization. The clone is used to (i) submit the operation coming from its mobile to the super clone (or central server), (ii) transform the operations of the other clones received from the super clone and (iii) handle its queues so that its state of the shared document is coherent to that of other clones. Note that the super clone serializes all operations according to a total order.

For instance, the clone in CloneDoc [1] maintains two states: *The pending queue* contains update operations received from its mobile that have already been applied to its local state, but not sent yet to the super clone (i.e. not yet serialized). *The committed queue* contains operations already ordered by the super clone.

When a clone receives an operation from its mobile, it applies it immediately over its local state, saves it locally and then sends it to the central server to be ordered. In the case of CloneDoc, the operation is saved in *the pending queue*.

[4] https://www.google.fr/intl/fr/docs/about/.

To serialize operations, two total ordering schemes may be enforced by the super clone [12]:

- *The implicit total ordering scheme*: it is based on a central server for broadcasting operations among collaborating clones where each clone sends local operations to the super clone via a FIFO (First In First Out) communication channel. Then, the super clone serializes operations and broadcasts them among all clones. The operation serialization order at the super clone *implicitly* forms a total order among all operations. This total ordering scheme is used in Google Docs.
- *The explicit total ordering scheme*: it is based on a special sequencer which does not broadcast operations but only generates continuous sequence numbers (tickets) for ordering operations. After generating a local operation, a collaborating clone requests the central sequencer for a sequence number to timestamp this operation. In this way, all operations are totally ordered by sequence numbers. This total ordering scheme is used in CloneDoc [1].

When the clone receives an ordered operation form the central server, two cases are possible:

- If the received operation is generated by another clone then, the clone transforms it over its precedent ordered operations (w.r.t the total order enforced by the super clone), applies it on its local state and sends it to its mobile. In the case of CloneDoc, the operation is transformed over *the committed queue*, then the resulted operation is applied on the local state of the clone and sent to the mobile device.
- If the clone received its own operation from the central server, the clone stores it locally and sends it to its mobile without executing it because it is already executed. In the case of CloneDoc, the clone extracts the received operation from *the pending queue*, adds it in *the committed queue* and sends it to its mobile.

Mobile-to-Clone synchronization. The mobile device and its clone are not physically the same. This leads to an inevitable delay in their communications which may introduce data inconsistency (as shown in Sect. 2.3). Therefore, an additional consistency protocol based on OT approach is deployed on mobile device to solve this problem. For example, a user applies several operations on her/his mobile to edit the local copy of the shared document in disconnected mode; these operations are logged in a local queue of the mobile device (a *pending queue* in the case of CloneDoc). At meanwhile, the clone can receive operations from other clones. When the mobile joins the cloud, its clone sends it the operations received and integrated from other clones. Then, the mobile transforms these operations over its local operations stored in its local queue.

Consider the scenario illustrated in Fig. 4 where User 1 exchanges operation O with User 2. First, User 1 executes immediately the operation O on the local copy and sends it to his clone namely Clone 1. Then, Clone 1 performs O, saves it in the pending queue and sends it to *SuperClone* in order to serialize it according to the used total ordering scheme. Next, Super Clone broadcasts the ordered operation O' to all the clones of the cloud, including Clone 1.

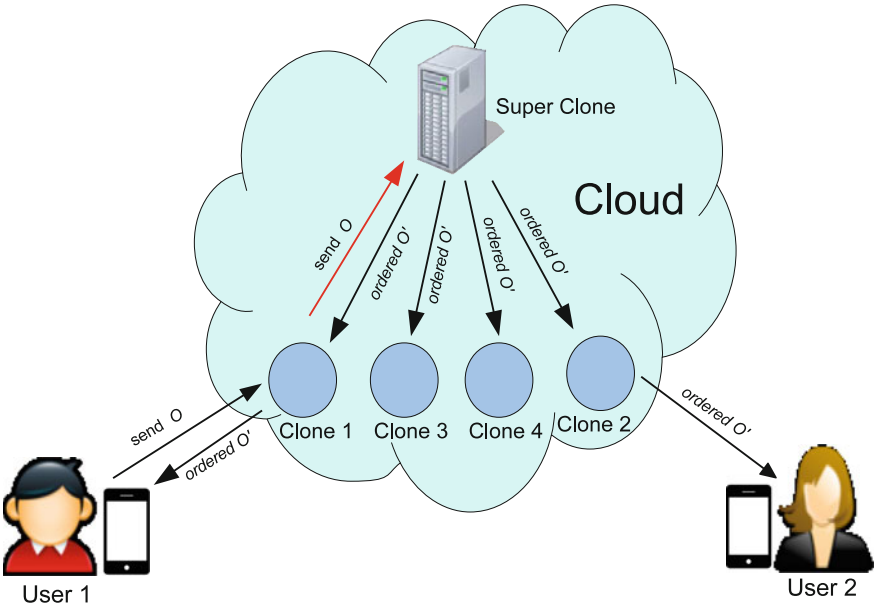


Fig. 4 Centralized synchronization

When Clone 2 receives the ordered operation O', it transforms it over its committed queue. Then, the clone executes the transformed operation over its local state and sends it to the mobile of User 2.

On the other side, when Clone 1 receives its own ordered operation O' from *SuperClone*, it extracts the operation from the pending queue, adds the ordered operation in the committed queue and sends it to the mobile of User 1.

The clones send back the ordered operation O' to their real devices (smartphones) to be transformed (if there are operations in pending queue) and executed (except in User 1) such that the copy of User 1 is coherent with the copies of the other users in the system.

3.2 Drawbacks

The strong dependence of different users (clones) to the coordination server (which plays a central and important role in the collaboration) can cause serious problems that may negatively affect the collaboration. Hereafter, we list some disadvantages of centralized concurrency control.

Failure. Based on central coordination server, mobile collaborative applications deployed on cloud environments are expected to suffer from bottlenecks and are more prone to faults. For example, if all clones running in the cloud keep making lot of updates to the super clone, the performance will be drastically downgraded at the

expense of responsiveness. Again, if the super clone fails (due to hardware failure or denial of service [13]), some clones will lose their updates. Moreover, a malicious user can flood the sequencer server with a great number of operations in order to make the server unavailable as long as possible.

Energy consumption. The collaborative software based on centralized concurrency control protocols, such as CloneDoc [1] and Google Docs, use an additional processing of OT on the mobile side in order to avoid inconsistency problem between the mobile and its clone as illustrated in Fig. 3. However, this will cause supplementary energy consumption.

Network traffic. The coordination based on super node (or central server) incurs more network bandwidth to receive and broadcast updates from/to clones respectively. Larger the number of users, more the network bandwidth is consumed. Therefore, this problem may lead to the loss of some operations of clones and consequently to data divergence.

Intention violation. The intention of an operation o means the effect which can be obtained by executing o on the document state from which o was generated. If one operation o is generated at clone i, then its intention should be preserved at any clone j (with $i \neq j$) regardless the transformations that o undergoes. Preserving users intention using OT approach is a hard task in collaborative editing applications. In the following, we present a scenario of violation of intention in Google Docs.

Google Docs allows users to modify and update the same document in the same time by using a central coordination server. Google Docs is based on Jupiter [14] which is a client-server collaborative system. The Jupiter server maintains multiple 2D state-spaces, one for every client. A state-space consists of a local dimension for operations generated by the corresponding client, and a global dimension for operations from all other clients. To avoid using 2D state-spaces, Google adapted Jupiter system [14] by adding a *Stop-and-Wait* protocol between the client and the server. As a consequence, a single 1D buffer at the server is sufficient to maintain all transformation states.

However, Google Docs inherits the main flaw of Jupiter system, namely intention violation [15]. As illustrated in Fig. 5, three users concurrently execute different operations on the same document that contains initially the state "A". User 1 performs operation $Ins(1, X)$ to add 'X' at the position 1. Simultaneously, User 2 executes operation $Ins(2, Y)$ to insert 'Y' at position 2 and User 3 performs $Del(1)$ to delete 'A'.

Then, the users send their operations to the central server in order to be transformed. The Google Docs server uses the reception order of operations to determine the priority among operations. Consequently, we might get different results depending on the reception order of operations.

In Fig. 5, we have three different reception orders which result in two divergent states "XY" and "YX" for the same document.

For example, If the server executes the delete operation o_3 first, the two insert operations o_1 and o_2 will be transformed to insert different characters at the same position in the document.

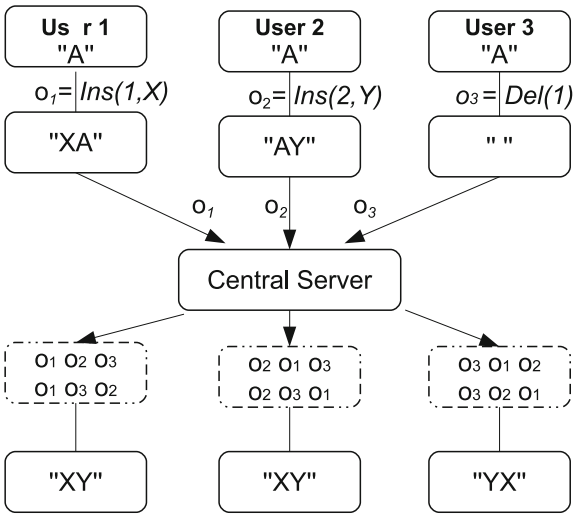


Fig. 5 Intention preservation scenario in Google Docs [15, 16]

Moreover, if we assume that the Google Docs server is potentially malicious (i.e. in the sense that it might diverge the states of different users at a given point of time), then it is very hard (or even impossible) for users to be aware about this problem and to recover back to the point of time when their views were consistent.

Security risks. The central server may be vulnerable to malicious side-channel attacks [17]. This may affect the synchronization process. For instance, BYOD (Bring Your Own Devices)[5] practice is increasingly becoming a global phenomenon. Indeed, it allows employees/students to bring personally owned mobile devices (laptops, tablets, and smartphones) to their workplace, and to use those devices to access privileged company information/education platforms. What happens if BYOD practice is used for accessing a collaborative application based on central coordination? It is clear that BYOD brings significant security risks. The users could be susceptible from attacks originating from compromised web sites that may contain harmful malware and compromise the proper functioning of the application.

4 Decentralized Concurrency Control

In this section, we give a general presentation of decentralized concurrency control scheme designed "à la Peer-to-Peer" and supporting an unconstrained collaborative work (without the necessity of central coordination). Using OT approach, synchronization of divergent copies is fulfilled automatically at each clone. To better present

[5] http://www.ibm.com/mobilefirst/us/en/bring-your-own-device/byod.html.

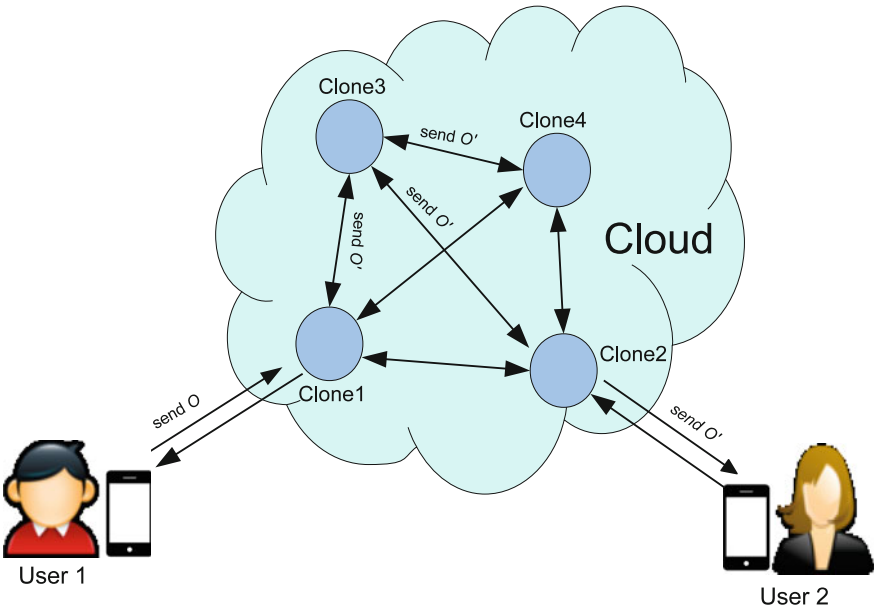


Fig. 6 Distributed synchronization

this decentralization, we describe the main components of OptiCloud [2, 3] as it is the best representative of decentralized concurrency control protocols combining mobile and cloud environments.

4.1 Principle

As illustrated in Fig. 6, the mobile collaborative application, based on decentralized coordination, provides a pure *peer-to-peer virtual private network* (without any server role assigned to some clone) platform where users can form ad-hoc groups based on their clones to achieve a common objective. It allows users to cooperate as follows (see Fig. 6): each user has a local copy of the shared document in her/his mobile and another copy of the same document in the clone; the user's operation O is locally executed in the mobile device and then is sent to its clone in order to be transformed (O' is the transformed form of O) and executed on other mobile devices (via their clones).

In [2, 3], a collaborative editing service is presented for manipulating the shared data, regardless of spatial and temporal constraints, where mobile users can edit collaboratively shared documents in peer-to-peer mode. The advantages of this model are (i) the availability of data anytime and anywhere, and (ii) the optimal use of mobile devices resources. In fact, the collaboration and communication tasks are

seamlessly turned on the cloud. Moreover, it is equipped with mechanisms to transparently manage the user departure, the arrival of new users joining the collaboration group.

Each clone has a local copy of the shared document S_C and its log L_C storing all updates received from other clones and updates generated from its mobile. The log L_R contains remote updates transformed against L_C, executed on S_C and in synchronization pending with the mobile. Updates sent by the mobile and generated locally in the clone are stored in log L_M. An index S_P of log L_C is used to indicate the last synchronization point between the clone and its mobile.

To preserve data consistency in peer-to-peer mode between mobile users, OptiCloud [2, 3] is constituted from two synchronization layers: The first layer ensures synchronization between clones (as back-end) and the second one consists in synchronizing the mobile with its clone (as front-end).

4.1.1 Clone-Clone Synchronization

Two events can be triggered in the clone: (i) receiving operations from mobile to be generated and integrated in the clone; (ii) receiving and integrating remote operations coming from other clones.

Generation of local operations. Once the clone receives operations from its mobile, it performs the following steps:

1. Computes the minimal execution context of each operation O received from mobile. An operation may depend on previous operations according to the execution order. Tracking this dependency inside a log enables to identify operations that must be executed on all clones according to the same order. The clone synchronization protocol uses a minimal dependency relation which is independent of the number of users, and accordingly, it is well suited for dynamic groups. In other words, instead of considering O as being dependent of all L_C operations, this step reduces this context by excluding as much as possible some operations of L_C to give the transformed operation O'. For more details, we can refer to [15, 18].
2. Sends O' to other clones and adds it in L_M which contains all operations (coming from the mobile) executed on L_C.
3. Determines the operations that are concurrent to O' in log L_R and calls the transformation component in order to get operation O'' that is the transformed form of O' according to the concurrent operations;
4. Applies operation O'' over its local state S_C and adds it to log L_R.

Integration of remote operations. When a clone receives a remote operation O from another clone, the integration of this operation proceeds by the following steps:

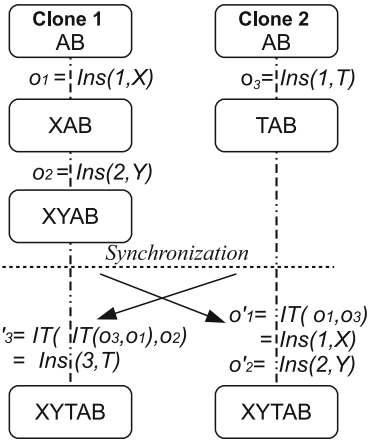


Fig. 7 Scenario of collaboration

1. From the log L_C, it determines the sequence *seq* of concurrent operations to O in order to transform O against *seq* and obtain O';
2. After added O' to the local log L_C, it determines from the log L_R the sequence seq' of operations that are concurrent to O';
3. It calls the transformation component in order to get operation O'' that is the transformation of O' according to seq';
4. It executes O'' on the current state and adds it to the local log L_R.

Illustrative example. Given two clones, Clone 1 and Clone 2 editing a shared document described in Fig. 7. Initially, each Clone has a copy that contains "AB". Two local insertion operations O_1 and O_2 have been executed by Clone 1. Concurrently, Clone 2 has executed another insertion operation O_3. The added characters are 'X', 'Y' and 'T' respectively.

There is a dependency relation between operations O_1 and O_2 in such a way O_1 must be executed before O_2 in all clones. This is due to the fact that their added characters are adjacent (positions 1 and 2) and created by the same clone (for more details see [18]). This dependency relation is minimal in the sense that when O_2 is broadcast to all clones, it holds only the identity of O_1 as it depends on directly. As illustrated in Fig. 7, the execution order is as follows.

At Clone 1, O_3 is considered as concurrent. It is then transformed against O_1 and O_2. The sequence $[O_1 = \text{Ins}(1, \text{X}), O_2 = \text{Ins}(2, \text{Y}), O'_3 = \text{Ins}(3, \text{T})]$ is executed and logged in Clone 1, where O'_3 results from transforming O_3 to include the effect of operations O_1 and O_2 (i.e. $O'_3 = \text{IT}(\text{IT}(O_3, O_1), O_2) = \text{Ins}(3, \text{T})$ using transformation function *IT* given in [18]).

At Clone 2, O_1 and O_2 are concurrent with respect to O_3. They must be transformed before being executed after O_3 according to their dependency relation. Thus, the following sequence is executed and logged in Clone 2: $O_3 = \text{Ins}(1, \text{T})$ and $O'_1 = \text{IT}(O_1, O_3) = O_1$ and $O'_2 = O_2$.

4.1.2 Mobile-to-Clone Synchronization

In OptiCloud [2, 3], the shared document is considered as a critical section between the mobile and its clone. Indeed, when the mobile tries to commit/synchronize w.r.t its clone, only one of them will have the exclusive right to access in synchronizing mode to its document copy. This distributed mutual exclusion protocol is achieved by the exchange of messages (i.e., token). Initially, the mobile device has the right to be the first to commit/synchronize with its clone. To ensure a safe synchronization between mobile and its clone, we use two tokens T_M and T_C:

- Token T_M gives the mobile state: (i) $T_M = 1$ means that mobile is editing its local copy; (ii) $T_M = 0$ indicates that mobile is synchronizing with its clone.
- Token T_C indicates the state of the clone: (i) $T_C = 1$ states clone is integrating remote operations received from other clones; (ii) $T_C = 0$ means that clone is synchronizing with its mobile.

Whatever where the exclusive access right is, the mobile device and its clone can edit independently their local copies. The clone continues to receive remote operations from other clones to integrate them later on the local state. At the meantime, the mobile user can work on her/his copy in unconstrained way. But, once she/he decides to synchronize with her/his clone, all local editing operations are sent to her/his clone and the exclusive access right is released to enable the clone to start the synchronization with the mobile device as illustrated in Fig. 8. Thus, the clone performs the operations issued by the mobile device on its local state, includes their effects by transformation in its local (and not yet seen by the mobile) operations, and sends the resulting (or transformed) operations to the mobile device in order to integrate them.

In the mobile side, OptiCloud uses just the local state and the *Log* that contains performed operations. Note that the mobile device does not perform any specific

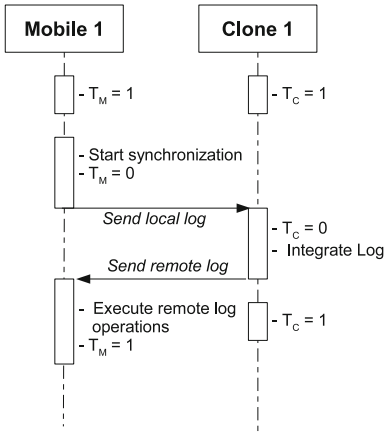


Fig. 8 Sequence diagram of synchronization process among mobile and clone

treatment; it updates only its state. The steps of synchronization between the mobile and its clone can be represented as follows:

Generation of local operations. If the mobile is in editing state (i.e. $T_M = 1$) then it creates operation o (e.g. $o = ins(1, X)$), executes this operation directly on its local state and adds o to its log. When the user wants to synchronize with her/his clone (i.e. $T_M = 0$ indicating that the mobile is in synchronizing state), the log is sent to the clone. The mobile cannot generate local operations up to the end of synchronization.

Reception of mobile operations by the clone. When the clone starts the synchronization process with its mobile, after receiving log L_M from the mobile, the clone generates locally each received operation (as previously explained in step ***Generation of local operations*** of Sect. 4.1.1). After integrating all operations coming from the mobile, the clone prepares the operations (not yet seen by the mobile) to be sent to the mobile. These operations are included in the interval from S_P to $|L_C|$ in log L_C, where S_p is the last synchronization point between the mobile and its clone and $|L_C|$ the length of log L_C. Each operation O in the interval $[S_P, |L_C|]$ inside log L_C is integrated over L_M as follows:

1. Defines the operations that are concurrent to O in the log L_M;
2. Transforms O according to the defined operations to result O';
3. Adds O' to the local log L_M;

Next, operation O' is added to a *Log*. Then, the *Log* is sent to the mobile in order to be applied on its local state.

The reception of operations from the clone. When the mobile receives operations from its clone, it applies them over its local state and makes $T_M = 1$ to indicate that it is available to generate local operations.

4.2 Drawbacks

Although the cloud-based mobile collaborative applications using distributed concurrency control avoid several flaws of the ones based on the centralized coordination server, they also have some weaknesses:

Energy consumption. Since the heavy computing tasks are delegated to the clone, it consumes more energy in the cloud in order to ensure communication among clones and also its mobile, compute transformation procedures to maintain data consistency and manage join and leave events of its mobile.

Access control. Ensuring a distributed access control to a shared data is a challenging problem in the cloud-based mobile collaborative application. The availability of the shared data in anytime and anywhere is one of the main requirements of collaborative applications, whereas access control looks to guarantee this availability only to users with proper authorization. Moreover, high responsiveness of local

updates is required. However, when adding an access control layer, high responsiveness is lost because every update must be guaranteed by some authorization coming from a distant user (such as central server) [19].

5 Related Work

The massive development of cloud technology encourages users to delegate their heavy computing treatments to cloud platforms. The mobile applications are the most concerned candidates to benefit from the huge computing power of Cloud to satisfy their constraints in terms of resources (battery life, date storage and application speed up).

Several approaches [20–22] have been proposed to offload parts of their heavy tasks to the cloud since execution in the cloud is considerably faster than the one on mobile devices and mobile-cloud offloading mechanisms delegate heavy mobile computation to the cloud.

SPORC [11] is a collaborative system where several users can edit shared documents using smartphones. To maintain consistency, it relies on operational transformation technique and a single server to give global order to concurrent user updates. However, SPORC exchanges a large number of messages between users and the server. Therefore, it is not well adequate to mobile devices due to short battery life.

Inspired from SPORC, CloneDoc [1] is a secure real-time which enables collaboration for mobile devices that are cloned in the cloud in order to lessen the heavy computing tasks of collaborative editing works on mobile devices. CloneDoc is implemented upon C2C platform [23] which is a distributed peer-to-peer platform for cloud clones of smartphones. It is based on Operational Transformation (OT) approach and a single server to enforce a continuous and global order to avoid the divergence of client's document view from the server. Unfortunately, a server failure could stop the collaboration between mobile devices. Moreover, CloneDoc needs additional treatment of OT on the mobile side to ensure convergence between the mobile and its clone.

rbTree-Doc [24] is a collaborative editor framework for cloud environments. It allows multiple users to share and edit online documents. The document is represented as Red-Black tree [25]. The user can download a part of a document to be updated from the cloud service, and this enables rbTree-Doc to reduce the amount of data that needs to be encrypted by focusing on the analysis of content that has been updated by the collaborative services. However, rbTree-Doc is based on a client-server model and uses a complex structure to represent the shared document, and this from one hand, cannot support a server failure situation and from the other hand, the complex structures are hardly treated in mobile devices.

Hermes [26] is a transparent approach to interoperate between heterogeneous collaborative editing services in the cloud. Users are enabled to use their familiar services to participate in the cross-cloud document collaboration. Hermes uses an OT driven approach to resolve conflicts and maintain data consistency for cross-cloud

document synchronization. Like Hermes, the collaborative editing service proposed in [2, 3] can be extended to manage the mobile collaboration on several clouds.

The platform presented in [2, 3, 27] is cloud service-based approach, and it is composed of two level systems. The first level, cloning engine, provides an automatic chain for preparing a dynamic platform that can contain multiple virtual private networks. Each network corresponds to a group of users and contains clones of their mobiles. It provides web services to manage users groups and creates clones of mobiles in the cloud. The second layer, OptiCloud, provides group collaboration mechanisms for editing shared documents in fully decentralized way. Unlike SPORC [11], CloneDoc [1] and rbTree-Doc [24], the platform of [2, 3] furnishes group collaboration mechanisms in real-time without any role assigned to the server. Procedures for maintaining consistency of shared documents are executed on the clone side.

6 Conclusion

Designing concurrency control for mobile collaborative applications is considered as a challenge, since mobile devices are constrained by insufficient resources which must be regarded when combining mobile and cloud environments.

In this chapter, we have presented the data consistency issues when mobile users are cloned in the cloud and they update simultaneously the shared data replicated in the mobile device and its clone. Two kinds of concurrency control scheme are described with their drawbacks. The first control uses central coordination server to maintain data consistency. As for the second one, it provides a synchronization mechanism in fully decentralized way.

As future research direction, it is interesting to add a new layer for security within mobile collaborative applications in the cloud. It consists in developing a protocol for managing distributed access rights and adding cryptographic mechanisms to ensure maximum security of shared resources.

Appendix

- *Replica*: is a copy of the shared data that can be modified at will by the user.
- *Data Consistency*: means that data values must be the same for all replicas when there is no updates in transit. This term is used to indicate that the system is able to reflect correctly the updates performed on a copy to all other copies of the shared data.
- *Data Concurrency*: means that many users can access simultaneously shared data to perform read and write operations.

- *Data Dependency*: an update operation applied on replica may depend on previously performed operations. In other words, the effect of such operation may be influenced by previous operations.
- *Clone*: in our case, the clone is a virtual machine Android X86[6] running in the cloud and has the same features as a physical mobile device.

References

1. Kosta, S., Perta, V.-C., Stefa, J., Hui, P., Mei, A.: Clonedoc: exploiting the cloud to leverage secure group collaboration mechanisms for smartphones. In: IEEE INFOCOM, Italy (2013)
2. Guetmi, N., Mechaoui, M.D., Imine, A., Bellatreche, L.: Mobile collaboration: a collaborative editing service in the cloud. In: SAC 2015, pp. 509–512, Spain (2015)
3. Mechaoui, M.D., Guetmi, N., Imine, A.: Lightweight and mobile collaboration across a collaborative editing service in the cloud. Peer-to-Peer Networking Appl. (2016)
4. Buchegger, S., Datta, A.: A case for p2p infrastructure for social networks—opportunities and challenges. In: WONS'09, pp. 149–156, Piscataway, NJ, USA (2009)
5. Vastardis, N., Yang, K.: Mobile social networks: architectures, social properties, and key research challenges. IEEE Commun. Surv. Tutorials **15**(3), 1355–1371 (2013)
6. Karam, A., Mohamed, N.: Middleware for mobile social networks: a survey. In: HICSS 2012, pp. 1482–1490, Maui, Hawaii (2012)
7. Borcea, C., Gupta, A., Kalra, A., Jones, Q., Iftode, L.: The mobisoc middleware for mobile social computing: challenges, design, and early experiences. In: MOBILWARE '08, pp. 1–27, ICST, Brussels, Belgium (2007)
8. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: SIGMOD Conference, pp. 399–407, Portland, USA (1989)
9. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality-preservation and intention-preservation in real-time cooperative editing systems. ACM Trans. Comput. Hum. Interact. **5**(1), 63–108 (1998)
10. Ressel, M., Nitsche-Ruhland, D., Gunzenhauser, R.: An integrating, transformation-oriented approach to concurrency control and undo in group editors. In: ACM CSCW'96, pp. 288–297, Boston, USA, November 1996
11. Feldman, A.J., Zeller, W.P., Freedman, M.J., Felten, E.W.: Sporc: group collaboration using untrusted cloud resources. In: OSDI, pp. 337–350, Vancouver, BC, Canada (2010)
12. Yi, X., Sun, C.: Conditions and patterns for achieving convergence in OT based collaborative editing systems. IEEE Trans. Parallel Distrib. Syst. **27**(3), 695–709 (2015)
13. Subashini, S., Kavitha, V.: Review: a survey on security issues in service delivery models of cloud computing. J. Netw. Comput. Appl. **34**(1), 1–11 (2011)
14. Nichols, D.A., Curtis, P., Dixon, M., Lamping, J.: High-latency, low-bandwidth windowing in the jupiter collaboration system. In: Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology (UIST), pp. 111–120, New York, USA (1995)
15. Abdessamad, I.: Conception Formelle d'Algorithmes de Réplication Optimiste. Vers l'Edition Collaborative dans les Réseaux Pair-à-Pair. Phd thesis, University of Henri Poincaré, Nancy, France, December 2006
16. Paull, D.: Google wave: intention preservation, branching, merging and TP2 (2010). http://www.thinkbottomup.com.au/site/blog/Google_Wave_Intention_Preservation_Branching_Merging_and_TP2
17. Baig, M.B., Fitzsimons, C., Balasubramanian, S., Sion, R., Porter, D.E.: Cloudflow: cloud-wide policy enforcement using fast vm introspection. In: IC2E '14, pp. 159–164, Washington, DC, USA (2014)

[6] http://www.android-x86.org/.

18. Imine, A.: Coordination model for real-time collaborative editors. In: Coordination Models and Languages, 11th International Conference, COORDINATION 2009, Lisboa, Portugal, June 9–12, 2009. Proceedings, pp. 225–246 (2009)
19. Cherif, A., Imine, A., Rusinowitch, M.: Practical access control management for distributed collaborative editors. Pervasive Mob. Comput. **15**, 62–86 (2014)
20. Fangming, L., Peng, S., Hai, J., Linjie, D., Yu., J., Di, N., Bo, L.: Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. IEEE Wirel. Commun. **20**(3), 1–10 (2013)
21. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: MobiSys, pp. 49–62, San Francisco, CA, USA (2010)
22. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: Clonecloud: elastic execution between mobile device and cloud. In: EuroSys, pp. 301–314, Paris, France (2011)
23. Kosta, S., Perta, V.C., Stefa, J., Hui, P., Mei, A.: Clone2clone (c2c): peer-to-peer networking of smartphones on the cloud. In: HotCloud, San Jose, USA (2013)
24. Yeh, S.-C., Ming-Yang, S., Chen, H.-H., Lin, C.-Y.: An efficient and secure approach for a cloud collaborative editing. J. Netw. Comput. Appl. **36**(6), 1632–1641 (2013)
25. Bayer, R.: Symmetric binary b-trees: data structure and maintenance algorithms. Acta Inform. **1**(4), 290–306 (1972)
26. Xia, H., Lu, T., Shao, B., Ding, X., Gu, N.: Hermes: on collaboration across heterogeneous collaborative editing services in the cloud. In: IEEE CSCWD, pp. 655–660, Hsinchu, Taiwan (2014)
27. Mechaoui, M.D., Guetmi, N., Imine, A.: Mobile co-authoring of linked data in the cloud. In: New Trends in Databases and Information Systems—ADBIS, pp. 371–381, Poitiers, France (2015)