A Survey on Operational Transformation Algorithms: Challenges, Issues and Achievements

[1]Santosh Kumawat
M.Tech Scholar
Poornima College of Engg.
Jaipur, Rajasthan, India

[2]Ajay Khunteta
Asst Prof. Dept. of CS
Poornima College of Engg.
Jaipur, Rajasthan, India

ABSTRACT

Consistency maintenance is one of the most significant challenges in the design and implementation of the systems, where a group of users is allowed to view and edit the same document at the same time. From geographically dispersed sites connected by communication networks Operational transformation (OT) is an established optimistic consistency control method in collaborative applications. In addition, they generally support two character-based primitive operations, insert and delete, in a linear data structure. This paper presents an integrative review of the evolution of operational transformation techniques. It discusses major issues, algorithms, achievements and remaining challenges. A comparative study is done of various algorithms of OT based on different parameters.

General Terms

Operational transformation (OT), optimistic consistency control method.

Keywords

Operational transformation (OT), Inclusion and Exclusion transformation, dOPT, TP2, etc

1. INTRODUCTION

Consistency maintenance is a fundamental issue in many areas of computing systems, including database systems [Bernstein et al. 1987], distributed systems [Birman et al. 1991], and groupware systems [Baecker 1992; Sun et al. 1998]. Real-time collaborative graphics editing systems allow a group of users to view and edit the same graphics document at the same time from geographically dispersed sites connected by communication networks. Consistency maintenance in the face of concurrent accesses to shared objects is one of the core issues in the design of these types of systems. Real time group editors (e.g., Grove and Reduce) are a category of distributed systems that allow a group of users to edit the same document collaboratively at the same time over a computer network, e.g., the Internet. They are frequently used as an effective research vehicle and model of a wide range of distributed interactive groupware applications that feature coordinated manipulation of shared data objects. Consistency maintenance is a critical and challenging issue in many interactive groupware applications that can be modeled as group editors. Due to the fact that human users are an integrated part of the system, there are specific requirements [1] [2]:

- High local responsiveness: A group editor should be as responsive as its single-user counterparts.
- Unconstrained interaction: The users should be able to edit any part of the shared document at any time in a way comparable to using single-user editors.
- Real-time communication: The users must be notified of each other's operations in a timely manner for mutual awareness and effective coordination.
- Consistency: Eventually the users must be able to see a converged version of the shared document that is consistent with their actual intentions.

Group editors usually replicate the shared document at each site. Each user's operations are executed on the local replica immediately without being blocked or delayed. Operations are then propagated to remote sites and concurrency control protocols are sought to repair inconsistencies. Traditional concurrency control methods, such as locking and serialization, are found unsuitable for interactive groupware applications. Moreover, concurrency control methods in traditional distributed systems in general only consider content consistency, i.e., that all replicas eventually converge, while overlooking intention consistency, i.e., that the converged content is what the users want.

Over the past decade operational transformation (OT) [4, 5, 7] has become an established method for consistency maintenance in group editors. Compared to alternative concurrency control methods such as locking and serialization, OT has been found uniquely promising in achieving convergence, causality and intention preservation without sacrificing responsiveness and concurrent work in Sun et al. [8]. The main property of OT, allowing users to edit any part of the shared data at any time, in particular matches the vision of Bentley and Dourish[3] of developing collaborative systems as a customizable collaboration medium. In light of their arguments, group editors with this property do not impose any constraint on how people use the technology. Users are allowed to use group editors for either asynchronous or synchronous editing. With the right" single- user features (e.g., formatting) and multi-user features, group editors appear capable of accommodating a variety of editing purposes, work styles, and processes [6].

2. OPERATIONAL TRANSFORMATION

OT[13, 14] is an optimistic consistency control method that lies in the heart of many collaborative applications such as group editors and Google Wave1. The method replicates the shared data at cooperating sites. Local operations are always executed as soon as they are generated by the user. Remote operations are transformed before execution to repair inconsistencies.

2.1 Inclusion and Exclusion Transformation

Sun et al [12] distinguishes inclusion transformation (IT) and exclusion transformation (ET) functions. The IT function requires that if O_1 is transformed against O_2, the effect of executing its transformed version O_1 '($O_{1'} = IT(O_1, O_2)$) on the document state that contains the effect of O_2 should be the same as the effect of executing O_1 on the document state that does not contain the effect of O_2. The ET function means that O_1 is transformed against O_2 ($O_{1'} = ET(O_1, O_2)$) in such a way that the effect of O_2 is effectively excluded from O_1.

2.2 Transformation Properties

Given two operations O_1 and O_2 let $O_{1'} = IT(O_1, O_2)$ and $O_2 = IT(O_2, O_1)$, transformation function IT is required to possess the following two properties [12]:

TP1: $O_1 \text{ o } O_{2'} = O_2 \text{ o } O_{1'}$

TP2: $IT(IT(O, O_1), O_2) = IT(IT(O, O_2), O_{1'})$

TP1 ensures that if O_1 and O_2, the effect of executing O_1 before O_2 is the same as executing O_2 before O_1. TP2 ensures that transforming any operation O along different paths will yield the same result. These two properties actually ensure that arbitrary communication order can lead to a consistent final document state. In other words, it is not necessary to conservatively enforce a global total order of operations because inconsistencies can always be repaired with operational transformation, if TP1 and TP2 can be satisfied

2.3 The dOPT Puzzle

The original dOPT algorithm failed in some cases. These cases were re-discovered by several research groups [9] [10] [11] [12] and are termed as the dOPT puzzle. Figure 1 illustrates a simple scenario of the dOPT puzzle. Consider two users, A and B, start a group editing session from the same document state "abcd". At site 1, user A executes operation O_1 =Del(0, "a") and then O_2 =Ins(2, "x"), yielding document state "bcxd".

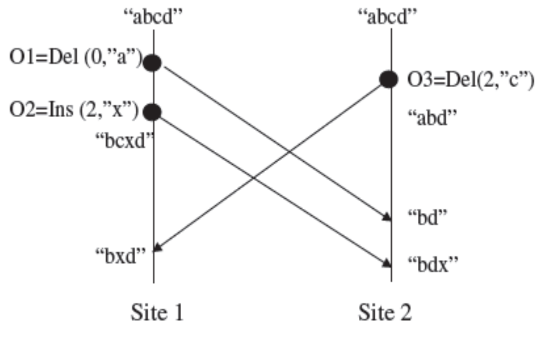

Fig. 1. A scenario of the dOPT puzzle.

.

At site 2, user B concurrently performs operation O_3= Del(2 , "c"), yielding "abd". According to the dOPT algorithm, a remote operation must be transformed against concurrent operations that have been executed locally. Then the transformed form is executed. So when site 1 receives $O_3$from site 2, (O_3 is transformed against O_1 and O_2 in order and becomes O_3=Del(1, "c")). After $O_{3'}$ is executed, the document state at site 1 becomes "bxd". At site 2, when O_1 arrives, it is transformed against O_3, O_1 is exactly the same as O_1 because its operation position precedes that of O_3. The document state of site 2 becomes "bd" after executing O_1. And then O_2 arrives and is transformed against O_3 which results in $O_2 = O_1$ because of the transformation rule (insert against delete) .After the execution of O_2, the document state of site 2 becomes "bdx", which is not consistent with the state of site 1.

The root of the dOPT puzzle is that, when operation O_2 is transformed against O_3 (at site 2, these two concurrent operations to be transformed are not originated from the same document state. The document state from which O_2 was generated includes the effect of O_1, while the document state from which O_2 was generated does not. So the position parameters of O_2 and O_3 are not comparable

2.4 The TP2 Puzzle

The In addition to the dOPT puzzle, another transformation puzzle was discovered by Sun et al. [16], which is illustrated by the scenario shown in Figure 2. In figure 2, three users start with the same initial document state "abc". User A performs an insert operation O_1 =Ins(2, "1") at site 1 and its document state becomes "ab1c". User B performs O_2 =Ins(1, "2") at site 2 and its document state becomes "a2bc". At site 3, user C performs O_3=Del(1, "b") and the document state becomes "ac". Those three operations are mutually concurrent with each other. When site 2 receives O_3, it is transformed against O_2 and becomes O_3 '=Del(2, "b"). After O_3 is executed, the document state of site 2 becomes "a2c". When O_1 arrives at site 2, it is transformed against O_2 and O_3 'in a sequel and becomes O_1'=Ins(2, "1"), which is the same as O_1 . After executing $O_{1'}$ the document state of site 2 becomes "a21c".

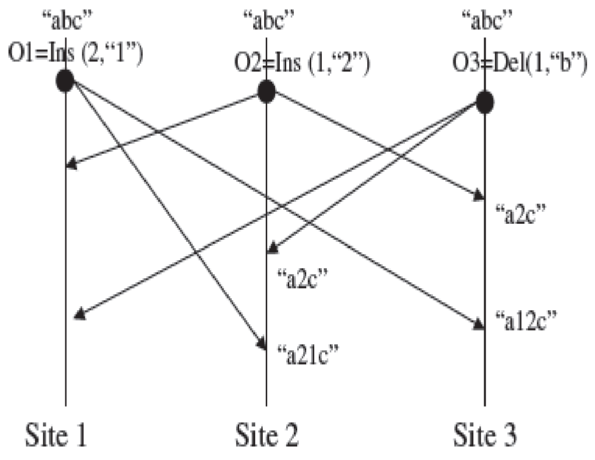

Fig. 2. The scenario of the TP2 puzzle.

At site 3, O_2 arrives first and is transformed against O_3. The transformation result O_2 is the same as O_2 and the document state becomes "a2c". When O_1 arrives, it is transformed against O_3 first and changed to $O_{1'}$=Ins(1, "1"). $O_{1'}$ is transformed against $O_{2'}$=Ins(1, "2"). Since the site id of O_2 ' is greater than one of O_1 ‘,O_1 ‘=Ins(1, “1”) is not modified. After O_1 ‘ is executed, the document state of site 3 becomes “a12c”, which is not consistent with the state of site 2.

The puzzle seems to be fixable by simply applying the following rule as proposed in [11]: when two insert operations have the same position parameter, the position of the operation with a larger site identifier will be shifted. Unfortunately, this quick fix works only in this case but fails in another similar scenario obtained by simply reversing the site identifiers of O_1 and O_2. The root of the problem is deeper than it appears and requires a more sophisticated solution than just considering the site identifier. The puzzle has a different nature from the dOPT puzzle.

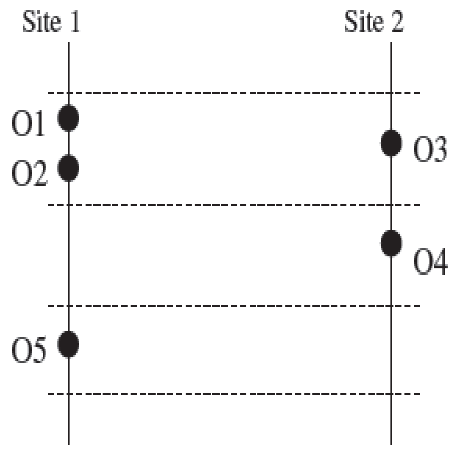

Fig. 3. Operations and time intervals.

Since TP2 fails in this case, we call it TP2 puzzle

3. ALGORITHMS

A plethora of OT algorithms have been proposed over the past two decades. There are two open challenges: First, most of them are developed under the framework of Sun, which includes an informal condition called "intention preservation". As a consequence, their correctness cannot be formally proved and counterexamples are often reported. Secondly, except for [11], all other OT algorithms only consider two character-based primitive operations. Although this simplification is theoretically acceptable, there is a practicality gap when applying those algorithms to real collaborative applications in which string based operations are common. The handling of string operations is very intricate.

3.1 dOPT

To achieve good responsiveness and avoid a single point of failure in the system, a replicated architecture & been adopted by GROVE the shined documents are replicated at the local storage of each participating site. An (update) operation is cxecuted on the local replica of the shared document immediately after its generation, then broadcast to remote sites for execution (after some delay and transformation). GROVE invented distributed Operation Transformation (dOPT) algorithm. GROVE’s solution consists of two components: one is the statevector timestamping scheme for ensuring the precedence property, and the other is the dOPT algorithm for ensuring the convergence property. The basic idea of the dOPT algorithm is that when an operation satisfies the precedence condition for execution, it is transformed against independent operations in the Log (which saves executes operations in the order of their execution) in such a way that executions of the same set of properly transformed independent operations in different orders produce identified document states, thus ensuring the convergence property.A sketch of the dOPT algorithm The transformation function T lies on the semantics of the editing operations and hence is application-dependent. The dOPT algorithm, however, is gentic and takes care of selecting operations for transformation and determining the transformation order. The basic control structure of the dOPT algorithms is simple: Given a causally ready operation O, the dOPT algorithm- the Log to transform O against any operation in the Log which is independent of then the transformed O, denoted as EO (i.e., the execution form of 0, is executed and saved in the Log.

3.2 adOPTed

The Jupiter consistency maintenance algorithm[18] was derived from the dOPT algorithm. The most interesting part of the Jupiter approach is the adaptation of the dOPT optimistic algorithm to an environment with multiple replicated clients sites plus one centrtized server site. In Jupiter, the shared documents are replicated at all cooperating client sites, which is the same as in GROVE. The difference is that the shared documents are also maintained at the central server and communications happen only between a client and the server.When an updating operation is generated on a client site, it is immediately executed at the local client site (for fast response to user actions), and then propagated to the central server. The server fit transforms the incoming operation if necesary, then executes the transformed operation on its copy of the shared document, and finally broadcasts the transformed operation to other client sites. Upon receiving an operation propagated from the central server, a client site may transform this operation if necessary, and then executes it on the local copy of the document. Convergence property is also getting satisfied here a bit.

The adOPTed algorithm added to the original dOPT algorithm a multidimensional interaction graph, which keeps track of all valid paths of transforming operations, and a double recursive function (similar in functionality to our GOT control algorithm) to determine which operations should apply the L-Transformation (similar to our Inclusion Transformation) against which others. If the L-Transformation functions could always satisfy the properties specified by Ressel et al. [1996], the adOPTed approach would be equivalent to our approach in the sense that the execution of the same set of operations on the same initial document by the two algorithms will produce the same outcome document the adOPTed algorithm works on an N-dimensional (where N is the number of cooperating sites) interaction graph containing all operations in various possible forms (i.e., the original, intermediate, and executed) in addition to a linear Log (the same as our history buffer) with operations in their original forms. The interaction graph provides a very useful model for visualizing the transformation relationship among original and transformed operations, but maintaining and searching a dynamically growing and potentially large N-dimensional graph at run time is inefficient and unnecessary (as proved by our approach). The adOPTed approach achieves both convergence and intention preservation at the application-dependent transformation algorithm level. The correctness of the adOPTed approach can be ensured by requiring LTransformation functions to guarantee the uniqueness of the labeling of vertices (for document states) and edges (for original /transformed operations) of the interaction graph.

3.3 IMOR

IMOR (Imine et al., 2003) ,[15] uses one new parameter in the insert operation, as insert(c; p; ip), where ip is the position of the operation relative to its generation state. When inclusively transforming two concurrent insertions whose positions tie, their ip values are compared, after which the ASCII codes of the characters are compared if there is still a tie. Intuitively, it may not make sense to compare the ip values of two concurrent operations if they are not defined on the same state. The nature of this problem resembles that of the dOPT puzzle (Sun and Ellis, 1998)

3.4 GOT

Although IT and ET functions defined in GOT[18] do not use explicit extra parameters, they use extra internal data structures to save information that helps break ties in some cases. are proposed to free TP2 by maintaining the same transformation path (total order of operations) at all sites every time an operation o is transformed. Although these algorithms can converge, those approaches are not always able to preserve the correct object order because they cannot prevent the loss of landmark characters in their transformation paths, without requiring TP1 and TP2, the GOT control algorithm, integrated with the undo/ do /redo scheme [17], is the only known solution for achieving both intention- preservation and convergence..

3.5 GOTO

The two additional post-conditions TP1 and TP2 can be employed to optimize the GOT control algorithm by reducing the number of IT/ET transformations. The optimized algorithm, named as GOTO [18] (GOT Optimized), resembles the Original GOT algorithm in handling the fit and the second cases. For the third case, the handling is different. In addition to performing transformations on the definition context of, we also perform transformations on the execution context. of O to make the two contexts equivalent. GOTO (Sun et al. 1998; Sun 2002) need extra memory for handling the so-called “lossy IT” problem. GOTO use function convert2HC(), which requires ET between an insert and a concurrent delete.

3.6 SOCT2

IT functions defined in SOCT2 (Suleiman et al. 1997) use explicit extra parameters, which take extra computation to derive. The transpose_bk function in SOCT2 (Suleiman et al. 1997; Suleiman et al. 1998) is also similar to our SWAP function. Note that SOCT2 only transforms two concurrent operations. SOCT2[19] implement an additional transformation, called Exclusion Transformation or Backward Transposition, which enables the order of execution of two consecutive operations to be changed without violating the user intention.

GOTO and SOCT2 : The space complexity of SOCT2 and GOTO are $O(|H|^2)$. The control procedure of SOCT2 and GOTO is more general under the established design framework of (Sun and Ellis 1998); the quadratic space complexity is derived basing on their provided transformation. Every time a remote operation is integrated, other algorithms (SOCT2, GOTO) call convert2HC() to transpose the whole history H. Hence the time complexities of all those algorithms are at least in the order of magnitude of $O(|H|^2)$.

3.7 SOCT3

Suppression of condition C2 requires the use of an unique global order precede S compatible with the causal order precede C. Moreover, in order to avoid to undo/redo operations the order of operations delivery must be consistent with the precede S order. It proposes to satisfy both constraints by using a sequencer to obtain a global and continuous order.

Local Execution, Broadcast and Reception of Operations in SOCT3[19]:A sequencer is an object which delivers continuously growing positive integer values, called timestamps. A timestamp is obtained through a call to function Ticket. The various methods of implementing a sequencer in a distributed system, namely circulating sequencer or replicated sequencer will not be discussed in this paper. Thanks to the Ticket function of the sequencer, each operation generated in the collaborative system is assigned a timestamp. The $precede_s$ order follows the order of the timestamps and we show below that it is compatible with the causal order precede.

3.8 SOCT4

In SOCT4 as in SOCT3, the operations are ordered globally using a timestamp given by a sequencer. They are then delivered on each site in this order thanks to the sequential reception. The originality of SOCT4[19] comes from the fact that forward transpositions that take into account concurrent operations are now made by the generator sites of the operations. This results in three major advantages:

a) The receiver site does not have to separate history any more ; thus backward transposition becomes unnecessary,

b) The received operation can be stored as it is in the history without further transformation,

c) State vectors are no longer needed. To achieve this, the broadcast of an operation must be deferred. More precisely, an operation generated on a site S is as usual executed locally without delay to satisfy the real−time constraint, but it is not broadcast until it has been assigned a timestamp and all the operations which precede it according to the timestamp order (i.e. precedeS) have been received and executed on site S. Moreover, before being broadcast, the operation is forward transposed with all concurrent operations, that is to say with operations received by S after its generation and preceding it in the global order.

3.9 TIBOT
Time interval based operational transformation algorithm (TIBOT) [17]that overcomes the various limitations of previous related work. Our approach guarantees content convergence and is significantly more simple and efficient than existing approaches. This is achieved in a pure replicated architecture by using a linear clock and by posting some constraints on communication that are reasonable for the application domain. TIBOT achieves the time complexity of O with the storage complexity significantly reduced, as compared to adOPTed, and the algorithm itself significantly simplified, as compared to GOTO. It use a linear time interval
based logical clock for the same purpose of causality preservation as the more complex vector clock approach in existing operational transformation algorithms. This increases system scalability in terms of accommodating late comers in a dynamic collaboration environment. Second, we solve the dOPT puzzle with a one- dimensional history buffer and Third, we solve the TP2 puzzle in a fully replicated architecture and without using ET (as compared to GOT or extra mechanisms.

3.10 WOOT & TTF
In WOOT (Oster et al. 2006a) and TTF (Oster et al. 2006b), Oster et al propose approaches that differ from the above. According to (Oster et al. 2005b), WOOT uses a model checker to prove convergence by verifying all cases that involve up to four sites and five characters, which deservers further work with regard to convergence. TTF uses a theorem prover to verify TP1 and TP2, which are sufficient conditions for convergence by (Ressel et al. 1996). In both works, they explain the concept of operation intention somewhat between the interpretation of (Sun et al. 1998) and our definitions of operation effects relation (Li and Li 2004; Li and Li 2007; Li and Li 2005). Nevertheless, they do not provide proofs with regard to their interpretation of intention preservation. It seems that our formalization of effects relation (Li and Li 2007) or admissibility in it [16] is compatible with and complementary to their approach of automated proofs. There is a potential that their approach and ours can leverage each other in future research.

3.11 LBT
LBT (Li and Li 2007) is the first work that builds special transformation paths (versus arbitrary paths). Each time a remote operation o is to be integrated, it[16] first transpose H into an H_C sequence $H_h \bullet H_c$. If o is a deletion, we inclusively transform o with H_c. If o is an insertion, however, we first transpose H_h into an ID sequence $H_{hi} \bullet H_{hd}$ and then transpose $H_{hd} \bullet H_c$ into another ID sequence $H_i \bullet H_d$. After that, o is first exclusively transformed with H_{hd} (the backward path) to exclude its effects and then inclusively transformed with $H_i \bullet H_d$ (the forward path) to include its effects. The correctness of IT is thus ensured: Since $H=H_{hi} \bullet H_i \bullet H_d$, when processing IT between insertion o with H_i, the landmark characters are all present. However, ET in the two transposition steps has to handle happened-before and concurrent operations ordered arbitrarily. The solution in LBT is to build ET-safe sequences by reordering the operations according to their effects relation. Consequently, ET is still very complicated.

3.12 SLOT
The SLOT transformation control algorithm is much simpler and more efficient than other algorithms such as GOTO. There are three other important advantages. Firstly, it is free of state vectors. State vectors are usually needed to capture concurrent relationships among operations, which have been achieved because the notification protocol ensures operations in OB and IB at the same site are concurrent. Secondly, it is free of ET transformation functions.

3.13 SDT
State difference based transformation (SDT)[15] approach which ensures convergence in the presence of arbitrary transformation paths. Our approach is based on a novel consistency model that is more explicitly formulated than previously established models for proving correctness. SDT is the first OT algorithm proved to converge in peer-to-peer group editors.It is able to ensure CSM consistency at the system level. In particular, it introduces a concept of operation effects relation and an approach to capture the correct effects relation between any operations. The performance of SDT in Li and Li (2005a): its worst-case time complexity is $O(n_3)$ and the expected time is $O(n_2)$.

3.14 ABT
Admissibility-based transformation (ABT) [16], that is theoretically based on formalized, provable correctness criteria and practically no longer requires transformation functions to work under all conditions. It requires only two correctness conditions, causality preservation and admissibility, that are formalized and provable. The new admissibility condition requires that the execution of every operation be admissible", i.e., not violating object relations that have been established by earlier admissible executions. Because convergence is implied by these two conditions, our consistency model does not include an explicit condition of convergence. Practically, it establishes a principled design methodology in which sufficient conditions of transformation functions are first identified and a suitable control procedure is then found to satisfy those sufficient conditions. This way, the control procedure and transformation functions are not separated as in previous works (Suleiman et al. 1998; Sun and Ellis 1998)—instead, they work synergistically in ensuring correctness; correctness of the algorithm can be easily proved without requiring the transformation functions to work in all possible cases. Due to the above properties of ABT, it is easier to develop OT algorithms and prove their correctness. In the ABT algorithm [16], the history H is maintained as an ID sequence $H_i \bullet H_d$ at every site and, before any operation o is propagated, the effects of H_d (the backward path), which contains all deletions that happened before o, have been excluded from o. As a result, when integrating a remote operation o, we only need to transpose H_i into an H_C sequence $H_{ih} \bullet H_{ic}$ and then 38 Du Li and Rui Li inclusively transform o with $H_{ic} \bullet H_d$ (the forward path). Hence the correctness of IT is ensured as simply as in LBT. However, ET is no longer required to work on arbitrary

paths: In function updateHL(), ET is only between an (insert or delete) operation and deletions that happened before it. In function updateHR(), ET is only between concurrent insertions. Therefore, the handling of ET in ABT is much simpler.

3.15 ABTS

The presented string wise ABTS algorithm is a significant extension to its character wise version ABT [21, 20].It supports string based primitive operations. The presented algorithm is the first of its kind with string wise operations and correctness formally proved. Specifically, when transforming two string wise operations, the algorithm is greatly complicated by the handling of position relations between the operation regions because operations may be split cascadingly during transformation. In it operations are stored in their execution order in the history H, the time complexity to integrate a remote operation is roughly O(IHI ^{2}) , The space complexity of the presented ABTS algorithm is trivially O(IHI).

In ABTS a history buffer H is maintained at each site which logs operations that have been applied to the data replica at that site. For correctness reasons [21, 20], H is maintained as a concatenation of of two subsequences, H_i and H_d' which record the executed insert and delete operations in their order of execution, respectively. That is, $H = H_i. H_d$. In addition, each site maintains RQ, a list of operations received from remote sites in their order of arrival. Each site j runs the following three concurrent threads:

Thread £ each time receives a local operation 0, applies it to the data replica, calls algorithm updateHL to update H and compute 0', a transformed version of 0, and propagates the resulting 0' to remote sites. Thread N receives remote operations from the network and appends them to RQ in their order of arrival. Thread R scans RQ for a remote operation 0 at a time that is causally- ready, i.e., all operations that happen before o have been executed at site j. Then algorithm update HR is called to update H and transform 0 into a version 0' that can be correctly executed in current state of site j. After that, 0' is executed on the data replica at site j.

3.16

The COT [22] algorithm the theory of operation context and, provide a new theoretical framework and uniformed solutions to both consistency maintenance and undo problems in distributed collaborative editing systems. With these results, we have achieved the goals to better understand and solve OT problems, reduce complexity, verify correctness, improve efficiency, and support the continual evolution of OT. The COT algorithm has two entries: the COT-DO entry for consistency maintenance (do) and the COT-UNDO entry for supporting undo. Operation context and context-based conditions are at the core of the whole COT algorithm. In the COT algorithm description, we use the context set representation C(O), rather than the context vector representation CV (O). This is because the context set representation is not only concise in expression but also directly implementable. Moreover, a document state DS is expressed as a set of original operations as well. By using original operation set expressions, we keep the COT algorithm independent of internal operation buffering schemes. When an operation O is propagated from the local site to remote sites, however, it is the context vector, not the operation set, that is actually piggybacked on O. Based on the information in CV (O), operations in C(O) can be easily determined from operations in DS.

4. CONCLUSION

Table 1 gives an overview of SOCT3, SOCT4, dOPT, adopted, GOT, GOTO and SOCT2 and many other algorithms. Many similarities exist among these algorithms regarding the techniques employed and we will take a closer look at the differences that make the originality of each one.

A relative comparison of a number of OT algorithms like dOPT, adOPTed, GOT, GOTO, SDT, SOCT2, SOCT3, SOCT4, ABT, ABTS ,SLOT is done relative to various parameters like various constraints like intention preservation, causality reservation, convergence ,in case of remote operations nature of communication and order of operation dispersion , and memory operations like order in memory or during integration operation type in memory and also other parameters like time complexity, space complexity, support for string handling, transformation functions and framework. A comparison with the existing algorithms concludes this article, and gives a synthetic overview of advantages and drawbacks of the different techniques implemented in each one. In this paper, we have reviewed a number of major operational transformation algorithms for consistency maintenance in real-time group editors, including the dOPT algorithm, the GOT algorithm, the GOTO algorithm, the SDT algorithm, the SCOT2, SCOT3, SCOT4 algorithm, the Jupiter algorithm, and the adOPTed algorithm, ABT algorithm, ABTS algorithm. In this conclusion section, we summarize the major achievements in the past 15 Years on the transformation-based consistency maintenance techniques and point out the major open issues for future exploration.

Table 1. Comparison of different algorithms(Parameters\Algorithms base table)

		dOPT	**adOPTed**	**GOT**	**GOTO**	**SOCT2**	**SOCT3**	**SOCT4**	**SDT**	**ABT**	**ABTS**
Correctness Criteria	**Intention preservation**	dOP Transformation	L−Transformation and multidimensional graph	Inclusion Transformation And Exclusion Transformation	Inclusion Transformation and Exclusion Transformation	Forward Transposition and Backward Transposition	Forward Transposition And Backward Transposition	Forward Transposition	Inclusion Transformation And Exclusion Transformation	IT and ET	IT and ET
	Causality preservation	State vectors	State vectors	State vectors	State vectors	State vectors	Timestamps	Timestamps	Vector timestamps	State vectors	Vector times tamps
	Convergence	Condition C1 (but convergence is not achieved)	Condition C1 And Condition C2	Non continuous global order and Undo/Redo	Condition C1 And Condition C2	Condition C1 And Condition C2	Condition C1 And Continuous global order	Condition C1 and Continuous global order	IT functions – two properties, TP1 and TP2, It along arbitrary transformation paths.	Satisfying TP1 and TP2	Admissibility preservation and Causality preservation
Property of operations of remote sites	**Nature of communication**	Immediate	Immediate	Immediate	Immediate	Immediate	Immediate (as soon as timestamp is assigned)	Deferred, in timestamp order	Every local operation is timestamped by the state vector of the state immediately resulted from its generation (and execution)	Vector timestamps	Causal Order
	Order of operation dispersion	Causal order	Causal order	Causal order	Causal order	Causal order	Continuous global order	Continuous global order	Causal Order - Reordering of history buffer before the integration of remote operations	Causal Order	Immediate
Storage	**Order in Memory**	Execution order	Several equivalent orders respecting the causal order	Global order (= execution order)	Optimized causal order	Optimized causal order	Continuous global order (≠execution order)	Continuousglobal order(≠executionorder)	Execution order	Execution order	Continuous global order

	During integration operations type in Memory	Executed operation	Received operation and some transformed operations	Executed operation	Executed operation	Executed operation	Transformed operation conforming to the timestamp order	Received operation	Executed Operation	Integrate every operation in an admissible manner and use the partial order	Received Operation
Time Complexity	**Time**	Consume more time	Less than dOPT	A bit less than adOPTed	$O(I\ H^{2}\ I)$ which is slower than ABT at least by some factor determined by the ratio of insertions in H	$O(I\ H^{2}\ I)$ which is slower than ABT at least by some factor determined by the ratio of insertions in H	A bit more than SCOT4	A bit more than ABT	$O(I\ H^{2}\ I)$ which is slower than ABT at least by some factor determined by the ratio of insertions in H Worst case $O(I\ H^{3}\ I)$ and expected time $O(I\ H^{2}\ I)$	$O(I\ H^{2}\ I)$	O(H)
Transformation Function		Condition C2	Condition C2	Condition C2	Condition C2	Condition C2	Global Order and C2	Global Order and C2	Condition TP1 and TP2 and IT and ET	Condition TP1 and TP2 and IT and ET	Condition TP1 and TP2 and IT and ET
Support for String handling		No	No	Yes	Yes	No	No	No	No	No	Yes
Framework		CC Framework	CC Framework	CC Framework	CC Framework	CC Framework	CC Framework	CC Framework	CCI Framework	ABT Framework	ABT Framework
Space complexity		More space	A bit less than dOPT	$O(I\ H^{2}\ I)$	$O(I\ H^{2}\ I)$	$O(I\ H^{2}\ I)$	A bit more than SCOT4	A bit more than ABT	A bit more than ABT	O(|H|)	O(|H|)

5. FUTURE WORK

Two types of consistencies one is syntactic consistency, which is concerned with whether all sites have the same view of the shared objects, regardless of whether the common view makes sense in the application context; and the other is semantic consistency, which is concerned with whether all sites have the same view of the shared objects, as well as whether the common view makes sense in the application context. There may exist many levels of syntactic consistency and semantic consistency in a particular application context. Previous work has mainly explored issues related to syntactic consistency. Particularly, the term intention defined and used in this paper has intention from the human user's perspective. This brings up interesting areas of research concerned with characterization and preservation of the human user's intentions in collaborative contexts, or group intentions. It may be infeasible for the system alone to automatically determine the human group intentions for different groups with divergent group goals. A lot of work is done to reduce time and space complexity .Still there is a scope to reduce time complexity and space complexity.

6. REFERENCES

[1] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In ACM SIGMOD'89 Preceedings, pages 399–407, Portland Oregon, 1989

[2] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality-preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 5(1):63–108,

[3] [3] R. Bentley and P. Dourish. Medium versus mechanism: Supporting collaboration through customization. In ECSCW'95 Proceedings, 1995.

[4] A. H. Davis, C. Sun, and J. Lu. Generalizing operational transformation to the standard general markup language. In ACM CSCW'02, pages 58{67, Nov. 2002.

[5] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In ACM SIGMOD'89 Preceedings, pages 399{407, Portland Oregon, 1989.

[6] S. Noel and J.-M. Robert. Empirical study on collaborative writing: What do co-authors do, use, and like. Journal of Computer Supported Cooperative Work, 13:63{89, 2004.

[7] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In ACM CSCW'98, pages 59{68, Dec. 1998.

[8] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 5(1):63{108, Mar. 1998.

[9] G. V. Cormack. A calculus for concurrent update. Technical Report CS-95-06, Dept. of Computer Science, University of Waterloo, Canada, 1995.

[10] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In ACM UIST'95 Proceedings, Nov. 1995.

[11] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In ACM CSCW'96 Proceedings, pages 288–297, Nov. 1996.

[12] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality-preservation, and intention- preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 5(1):63–108, Mar. 1998.

[13] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In Proceedings of the ACM SIGMOD'89 Conference on Management of Data, pages 399-407, Portland Oregon, 1989.

[14] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In Proceedings of the ACM Conference on Computer-Supported Cooperative Work, pages 59-68, Dec. 1998.

[15] D. Li and R. Li. An approach to ensuring consistency in peer-to-peer real-time group editors. Computer Supported Cooperative Work: The Journal of Collaborative Computing, 17(5-6):553-611, Dec. 2008.

[16] D. Li and R. Li. An admissibility-based operational transformation framework for collaborative editing systems. Computer Supported Cooperative Work: The Journal of Collaborative Computing, Aug. 2009. Accepted.

[17] R. Li, D. Li, and C. Sun, "A Time Interval Based Consistency Control Algorithm for Interactive Groupware Applications," Proc. IEEE Int'l Conf. Parallel and Distributed Systems (ICPADS '04), pp. 429-436, July 2004.

[18] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In Proceedings of the ACM Conference on Computer-Supported Cooperative Work, pages 59-68, Dec. 1998.

[19] N. Vidot, M. Cart, J. Ferrie, and M. Suleiman, "Copies Convergence in a Distributed Realtime Collaborative Environment," Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '00), pp. 171-180, Dec. 2000

[20] D. Li and R. Li. An admissibility-based operational transformation framework for collaborative editing systems. Computer Supported Cooperative Work: The Journal of Collaborative Computing, Aug. 2009. Accepted.

[21] R. Li and D. Li. Commutativity-based concurrency control in groupware. In Proceedings of the First IEEE Conference

[22] on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom '05), San Jose, CA, Dec. 2005.

[23] D. Sun and C. Sun. Context-based operational transformation in distributed collaborative editing systems. IEEE Transactions on Parallel and Distributed Systems, 20(10):1