Sequence Related IT Functions for String Based Update Operations by New Operational Transformation Algorithms for Wide-Area Collaborative Applications

[1]Santosh kumawat
Mtech Scholar
Poornima College of Engineering
34, Meera Kutir, Jain Mohalla,
Main Bazar, Sanganer, Jaipur-302029

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

***Abstract*— Operational transformation (OT) is an established optimistic consistency control method in collaborative applications. This approach requires correct transformation functions. In general all OT algorithms only consider two character-based primitive operations and hardly two or three of them support string based two primitive operations, insert and delete. In our earlier paper [1] we have proposed new algorithms that consider first time in history new string operations that are update for atomic string operations in addition to primitive operations like insert and delete. In this paper we have proposed new algorithms LocalU and RemoteU for handling local and remote update operations of strings for both atomic and sequential operations. These algorithms first time in history are handling sequential update string operations. These algorithms satisfy correctness criteria like causality preservation and admissibility preservation. These algorithms are for sequential update operations but have synchronization with existing primitive operations like insert and delete also. It also handles overlapping and splitting of operations when concurrent operations are transformed. These algorithms can be applied in a wide range of practical collaborative applications.**

Key words: Operational transformation, transformation functions, update string sequential operations, collaborative applications.

I. INTRODUCTION

Operational Transformation (OT) was originally invented for consistency maintenance in plain-text group editors [8]. In over 20 years, OT has evolved to support an increasing number of applications, including group undo, group-awareness, operation notification and compression, spreadsheet and table-centric applications, HTML/XML and tree-structured document editing, word processing and slide creation, transparent and heterogenous application-sharing and mobile replicated computing and database systems. To effectively and efficiently support existing and new applications, it must continue to improve the capability and quality of OT in solving both old and new problems. The soundness of the theoretical foundation for OT is crucial in this process. One theoretical underpinning of all existing OT algorithms is causality/concurrency causally related operations must be executed in their causal order; concurrent operations must be transformed before their execution.

Operational Transformation (OT) [3] is an established optimistic consistency control method in collaborative applications network. Consistency control in this environment must not only guarantee convergence of replicated data, but also attempt to preserve intentions of operations. Fast local response and timely group awareness are accepted performance metrics in group editors. In general optimistic consistency control on linear data structures is done. In this context a family of optimistic concurrency control algorithms called OT has been well established. OT allows to build real time groupware tools by correct transformation functions.

The objective of a collaborative environment [10] is to facilitate team working and, in particular, to enable a group of persons to manipulate shared objects, and modify them in a coherent manner. Moreover, an integrated set of schemes and algorithms, which support the proposed consistency model, are devised and discussed in detail. In particular, it have contributed (1) a novel generic operation transformation control algorithm for achieving intention preservation in combination with schemes for achieving convergence and causality preservation and (2) a pair of reversible inclusion and exclusion transformation algorithms for string wise operations for text editing

A plethora of OT algorithms have been proposed over the past two decades. Most of OT algorithms are developed under the framework of Sun et al [11], which includes an informal condition called "intention preservation". As a consequence, in general their correctness cannot be formally proved. In general all OT algorithms only consider two character-based primitive operations and hardly two or three of them support string based two primitive composite operations, insert and delete. In real collaborative applications in which string based operations are common. The handling of string operations is very intricate, as confirmed in [11]. So

there is a open challenge to handle more string composite operations.

To address the above challenges, this paper proposes two OT algorithms RemoteU and LocalU. It is based on the ABT framework [12, 13] which formalizes two correctness condition, causality and admissibility preservation. Causality preservation needed whenever an operation o is executed at a site, all operations that happen before o must have been executed at that site. Conceptually, admissibility requires that the execution of every operation not violate the relative position of effects produced by operations that have been executed so far. In general the ABT framework algorithms can be formally proved. The new proposed algorithms first time are handling string operations like update for atomic and composite operations both in addition to primitive operations insert and delete. Earlier our paper [1] first time in history has given algorithms for atomic update operations for strings and now in this paper we are proposing algorithms what handle both atomic and sequential update operations for strings, first time in history. It handles overlapping and splitting of operations when concurrent operations are transformed. These algorithms can be applied in a wide range of practical collaborative applications that require atomic string operations. Moreover, the design of these algorithms will provide a new starting point when extending OT algorithms to support composite and block operations that semantically must be applied together, such as cut-paste and find-replace.

A. OT Functions- Inclusion and Exclusion Transformation

OT functions used in different OT systems may be named differently, but they can be classified into two categories.

One is **Inclusion Transformation** (or Forward Transformation): IT(O_a, O_b) or T(op_1,op_2), which transforms operation O_a against another operation O_b in such a way that the impact of O_b is effectively included and the other is **Exclusion Transformation** (or Backward Transformation) : ET (O_a, O_b) or T^{-1}(op_1,op_2), which transforms operation O_a against another operation O_b in such a way that the impact of O_b is effectively excluded.

II. Background and Related Work

The philosophy of O'I' is to avoid operation overwriting so as not to lose user interaction results. The objective of a collaborative environment [15] is to facilitate team working and, in particular, to enable a group of persons to manipulate shared objects, and modify them in a coherent manner.

A. System Model and Notations

A number of collaborating sites is there in a system. The shared data is replicated at all sites when a session starts. Local operations are executed immediately and for local responsiveness, each site submits operations only to its local replica. In the background, local operations are propagated to remote sites. The shared data is like a linear string of atomic characters. Objects are referred to by their positions in the string, starting from zero. It consider two only primitive operations, namely, insert(p, s) and delete(p, s), which insert and delete a string s at position p in the shared data, respectively. Any operation o has attributes like o.id is the unique id of the site that originally submits o; o.type is the operation type which is either insert or delete; o.pos is the position in the shared data at which o is applied; o.str is the target string which the operation inserts or deletes. For a operation o, o.pos is always defined relative to some specific state of the shared data.

In the following table1 from [3] general notations of operation are summarized.

TABLE I. A Summary Of Main Notations

Notation	Brief Description
o.id	the id of site that originally generates o
o.type	the operation type of o, either ins or del
o.pos	the position of o relative to the data model
o.str	the string inserted or deleted by o
$o_1 \rightarrow o_2$	o_1 happens before o_2
$o_1 \parallel o_2$	o_1 and o_2 are concurrent
$o_1 \sqcup o_2$	o_1 and o_2 are contextual equivalent
$o_1 \mapsto o_2$	o_1 and o_2 are contextually serialized
$[o_1, o_2]$	an ordered list of two operations
$< o_1, o_2 >$	a 2-operation sequence in which $o_1 \mapsto o_2$
$\|L\|$	the number of objects in list/sequence L
$L_1 \cdot L_2$	a concatenated list/seq of two lists/seqs

To support string wise transformation, we need to introduce a few more notations. Given any string s, notation |s| is the number of characters in s. If 0 <= i<j <= |s|, notation s[i:j] returns a substring of s starting from position i to position j -1. If j is not specified, s [i:] returns a substring from i to the end. For example, let s="abc", then |s|=3 and s[0:2]="ab" and s[1:]="bc".

B. Literature Survey

Research on real-time group editors in the past decade has invented an innovative technique for consistency maintenance, under the name of operational transformation, which was pioneered by the GROVE Since then, several search groups have independently extended the operational transformation technique in their design and implementation of these types of systems. The limitation of causality had caused correctness problems from the very beginning of OT history. The dOPT algorithm was the first OT algorithm based on concurrency relationships among operations: a pair of

operations is transformable as long as the operations are concurrent.

Major representatives in this area include the REDUCE (REal-time Distributed Unconstrained Cooperative Editing) system, the Jupiter system, and the adOPTed algorithm. And then a new optimized generic operational transformation control algorithm get proposed.

Algorithms like aDOPTed, GOTO are used to maintain the consistency of shared data.

Operational transformation control algorithm SLOT for concurrency control, is significantly simpler and more efficient than existing algorithms. Furthermore, it is free of state vectors, free of ET transformation functions, and free of the TP2 transformation condition.

COT (Context-based OT) algorithm and the theory of operation context is capable of capturing essential relationships and conditions for all types of operation in an OT system; it provides a new foundation for better understanding and resolving OT problems.

To ensure the convergence of the copies while respecting the user intention, it have proposed two new algorithms, called SOCT3 and SOCT4.

A novel state difference based transformation (SDT) approach which ensures convergence in the presence of arbitrary transformation paths.

It proposes an alternative framework, called admissibility-based transformation (ABT), that is theoretically based on formalized, provable correctness criteria and practically no longer requires transformation functions to work under all conditions. Compared to previous approaches, ABT simplifies the design and proofs of OT algorithms.

Next it is having ABTS for string handling. First, it is based on a recent theoretical framework with formal conditions such that its correctness can be proved. Secondly, it supports two string based primitive operations and handles overlapping and splitting of operations. As a result, this algorithm can be applied in a wide range of practical collaborative applications.

III. ALGORITHMS

A history buffer Seq is maintained at each site which logs operations that have been applied to the data replica at that site. For getting better correctness Seq is maintained as a concatenation of three sequences SeqU, SeqIns and SeqDel which record the executed update, insert and delete operations in their order of execution, respectively. That is Seq=SeqU.SeqIns.SeqDel. 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 o, applies it to the data replica, calls algorithm LocalU to update Seq for update operations and compute o', a transformed version of o, and propagates the resulting o' 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 o at a time that is causally ready, i.e., all operations that happen before o have been executed at site j. Then algorithm RemoteU is called to update Seq and transform o into a version o' that can be correctly executed in current state of site j. After that, o' is executed on the data replica at site j.

Note here we are appending our new sequence SeqU in stating of Seq before SeqIns and SeqDel, so it is not affecting the appending process of new coming insert/ delete operations to the existing Seq, so what algorithms we are having in history for insert/ delete operations the same can get applied in the proposed scenario also that's why in this paper we are proposing new algorithms only for newly proposed update operation for strings.

A. Algorithm LocalU

Algorithm1 LocalU(o): o'

1. SeqID←SeqIns.SeqDel
2. if o.type=update then
3.(o', SeqID')←swapUSqID(SeqID, o)
4. Seq←SeqU. o'. SeqID'
5.endif
6. return o'

Seq is maintained as Seq=SeqU.SeqIns.SeqDel, so a new local update operation o_u must append to SeqU and insertion and deletion in SeqID where insertion will append to SeqIns and deletion will append to SeqDel. All operations executed on the local data replica are in Seq, the new local operation o is defined in current state of shared data. All operations in Seq happen before o (Seq→o). Here Seq and o are contextually serialized(or Seq→o). So o_u cannot directly append to SeqU due to the presence of SeqID because SeqU→o_u does not holds. We solve this problem by computing o_u', some version of o_u , such that SeqU→o_u'. This is achieved by swapping SeqID and o_u. Before the swapping SeqID→o_u holds and after swapping we get SeqID' and o_u' such that o_u'→ SeqID' holds. So o_u' can be appended to SeqU.

Based on above explanation in algorithm LocalU if the new local operation o is an updation, we swap it with SeqID and append the resulting o' to SeqU. Then we update the history to SeqU.o'.SeqID'. The resulting o' is returned and will be propagated to remote sites. The algorithm swapUSqID will be explained later.

B. Algorithm RemoteU

Algorithm RemoteU append o to SeqU where o is a remote operation and it return o' such that o' can be executed

- Santosh Kumawat is M.Tech Scholar in CS/IT Deptt of Poornima College of Engineering, Jaipur, Rajasthan, India.
- Ajay Khunteta is Asst Prof in CS/IT Deptt of Poornima College of Engineering, Jaipur, Rajasthan, India.

in current state. In it we are transposing SeqU in two contextually serialized sequences SeqUh and SeqUc such that SeqU= SeqUh.SeqUc where SeqUh contain all operations in SeqU that happen before o and SeqUc contain all operations in SeqU that are concurrent with o. Then history Seq is equal to SeqUh.SeqUc.SeqIns.SeqDel

Algorithm 2 RemoteU(o): o'

1. (SeqUh,SeqUc)←transposeHC(SeqU,o)
2. o'←SITOSq(o, SeqUc)
3. SeqID←SeqIns.SeqDel
4. if o.type=update then
5. SeqID'←SITUSqID(SeqID, o')
6. Seq← SeqU. o'. SeqID'
7. endif
8. return o'

As in Algorithm 2, we specify function RemoteU based on the above discussions. In line 1, it first transposes SeqU into two contextually serialized subsequences, SeqUh,SeqUc, by calling algorithm transposeHC. In line 2, it calls algorithm SITOSq to get o' by transforming o with SeqUc. Then in line 3, it assigns SeqIns.SeqDel to SeqID. If o is an updation, it transforms SeqID to incorporate the effect of o' in line 5. After that in line 6, o' is added between SeqU and the resulting SeqID'.

Algorithm transposeHC is already well-understood [4, 5, 6, 7] and here omitted. The IT function, SITOSq from our paper[2] and SITUSqID , will be explained in later Section .

C. Basic Functions

- *Basic IT Functions :*

In the most basic form, function IT(o_1,o_2) transforms a primitive operation o_1 with another primitive operation o_2 and outputs result o_1'. The output result can be a composite operation or atomic operation. According to [14], the precondition of IT(o_1, o_2) is $o_1 U o_2$ and the postcondition is $o_2 \rightarrow o_1'$. From[1] basic IT functions are ITUI and ITUD for atomic update operations where ITUI Algorithm transforms operation update o_1 with another operation that is insertion o_2 to incorporate the effects of o_2 in o_1 and ITUD transforms operation update o_1 with another operation that is deletion o_2 to incorporate the effects of o_2 in o_1.

- *Basic Swap Functions*

The basic swapping function for swapping two operations. Given two operations o_1 and o_2, where $o_1 \rightarrow o_2$, function swap(o_1, o_2) transposes them into o_1' and o_2' such that $o_2' \rightarrow o_1'$.The precondition of swap(o_1, o_2) is $o_1 \rightarrow o_2$.

From[1] basic swap functions are swapUI and swapUD where swapUI swaps an updation o_1 and an insertion o_2 and swapUD swaps an updation o_1 and an deletion o_2. Algorithm swapUI and swapUD is to swap update operation on string with other primitive operations like insertion and deletion on strings.

D. Sequence Related Functions

Algorithm 3 SITUSqID

SITUSqID(Sq, o): Sq'
1. op←o
2. SeqIns.SeqDel←Sq
3. olins←getSubOpList(SeqIns)
4. oldel←getSubOpList(SeqDel)
5. ol1←[] ol2←[]
6. for (i=0; i< |olins|; i++)do
7. o1←op
8. op←ITUI(op, olins[i])
9. or ←ITIU(olins[i], o1)
10. ol1←ol1.(or.sol)
11. endfor
12. for (i=0; i< |oldel|; i++)do
13. o2←op
14. op←ITUD (op, oldel [i])
15. or ←ITDU(oldel[i], o2)
16. ol2←ol2.(or.sol)
17. endfor
18. SeqIns'←combineSubOpList(ol1)
19. SeqDel'← combineSubOpList(ol2)
20. Sq'← SeqIns'.SeqDel'
21. return Sq'

We specify function SITUSqID(Sq, o) for transforming a sequence Sq with an operation o to incorporate the effects of o into every operation in Sq where SeqIns.SeqDel←Sq means sq consist of at left sequence SeqIns for insertion and then SeqDel for deletion in a linear fashion. Then we call functions ITUI, ITIU to inclusively transform operation o with SeqIns for insertion operations sequence and get transformed sequence ol1 and ITUD, ITDU to inclusively transform operation o with deletion operations sequence SeqDel and get transformed sequence ol2. Finally we merge all suboperations in ol1 into SeqIns' and ol2 into SeqDel' and return Sq' where Sq'←SeqIns'.SeqDel'

Algorithm 4 swapUSqID

swapUSqID(sq, o):(o', sq')
1. o'←o
2. SeqIns. SeqDel←sq
3. ol1← getSubOpList(SeqIns)
4. ol2← getSubOpList(SeqDel)
5. for(i=|ol1 |-1;i>=0;i--) do
6. (o', ol1[i])←swapUI (ol1[i], o')

7. endfor
8. for(i=|ol2 |-1;i>=0;i--) do
9. (o', ol2[i])←swapUD(ol2 [i], o')
10. endfor
11. SeqIns'←combineSubOpList(ol1)
12. SeqDel'← combineSubOpList(ol2)
13. sq'← SeqIns'.SeqDel'
14. return (o', sq')

As in Algorithm 4, function swapUSqID(sq, o) transposes a sequence sq of insertion and deletion with an updation o, where sq ← SeqIns. SeqDel, means sq consist of at left sequence SeqIns for insertion and then SeqDel for deletion in a linear fashion. Before swapping there is sq→o and after swapping there is o'→ sq'. We first flatten sq by collecting all sub-operations of sq in list ol1 and ol2 for sub lists SeqIns and SeqDel respectively. Then we call the specified function swapUI and swapUD to transpose every operation in ol1 and ol2 respectively with o from right to left. Finally we merge all suboperations in ol1 into SeqIns' and ol2 into SeqDel' and return the resulting sequence as sq' and o' where sq'← SeqIns'.SeqDel' and o' transposed form of operation o.

Functions swapUD, swapUI, ITUI and ITUD from our paper[1]. Functions getSubOpList and combineSubOpList from [3].

IV. CONCLUSION

In this paper we have proposed new optimized generic operational transformation algorithms that first time in history consider string operation update for sequence related string operations in addition to atomic operations. In past first time in history we have proposed algorithms in [1] for atomic update string operations and now in this paper we are proposing algorithms for update operations that support sequence related string operations. It also support existing primitive operations like insert and delete. Also since SeqU is in starting of Seq where Seq←SeqU.SeqIns.SeqDel, so it is not affecting the earlier algorithms for insertion and deletion. The new algorithms for update operations is also supporting earlier algorithms for insertion and deletion in history.

Most of OT algorithms are developed under the framework of Sun et al [11], which includes an informal condition called "intention preservation". As a consequence, in general their correctness cannot be formally proved. In general all OT algorithms only consider two character-based primitive operations and hardly two or three of them support string based two primitive operations, insert and delete.

To address the above challenges, this paper proposes a novel OT algorithm. It is based on the ABT framework [12, 13] which formalizes two correctness condition, causality and admissibility preservation. In general the ABT framework algorithms can be formally proved. The new proposed algorithms first time in history are handling string operations like update for sequential operations in addition to primitive operations insert and delete and handles overlapping and splitting of operations when concurrent operations are transformed. These algorithms can be applied in a wide range of practical collaborative applications that require string operations.

This paper proposed new algorithm like RemoteU and LocalU for both sequential and atomic string operations.

Moreover, the design of these algorithms will provide a new starting point when extending OT algorithms to support composite and block operations that semantically must be applied together, such as cut-paste and find-replace.

A. A. Future Work

There is a lot of efforts needed to preserve intention preservation and also to preserve semantic consistency and syntactic consistency. There is still scope to extend the support to other composite operations of string handling and char handling. Also it can support other better data structures also. A lot of work is done to reduce space complexity and time complexity. Still there is a scope to reduce space complexity and time complexity.

REFERENCES

[1] [1]Santosh Kumawat, [2]Ajay Khunteta Supporting More String Based Operations by New Operational Transformation Algorithms in Real-Time Cooperative Editing Systems International Journal of Engineering Science and Technology (IJEST) ISSN: 0975-5462 Volume 2 Issue 7, July 2010

[2] [1]Santosh Kumawat, [2]Ajay Khunteta New Optimized Generic Operational Transformation Consistency Control Algorithms Supporting String Operations in Collaborative Applications JOURNAL OF COMPUTING, VOLUME 2, ISSUE 7, JULY 2010, ISSN 2151-9617

[3]ABTS: A Transformation-Based Consistency Control Algorithm for Wide-Area Collaborative Applications Bin Shao , Du Li , Ning Gu . IEEE Paper published in 2009

[4] 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.

[5] R. Li and D. Li. Commutativity-based concurrency control in groupware. In Proceedings of the First IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom '05), San Jose, CA, Dec. 2005.

[6] M. Suleiman, M. Cart, and J. Ferrie. Concurrent operations in a distributed and mobile collaborative environment. In
IEEE ICDE '98 International Conference on Data Engineering, pages 36-45, Feb. 1998.

[7] 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.

[8] D. Sun, S. Xia, C. Sun, and D. Chen, "Operational Transformation for Collaborative Word Processing," Proc. ACM Conf. Computer- Supported Cooperative Work (CSCW '04), pp. 162-171, Nov. 2004.

[9] M. Suleiman, M. Cart, and J. Ferrie. Concurrent operations in a distributed and mobile collaborative environment. In IEEE ICDE '98 International Conference on Data Engineering, pages 36-45, Feb. 1998.

[10] M. Suleiman, M. Cart, and J. Ferrie. Concurrent operations in a distributed and mobile collaborative environment. In IEEE ICDE '98 International Conference on Data Engineering, pages 36-45, Feb. 1998.
[11] 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.
[12] R. Li and D. Li. Commutativity-based concurrency control in groupware. In Proceedings of the First IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom '05), San Jose, CA, Dec. 2005.
[13] 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.
[14] 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.
[15] G. Oster, P. Urso, P. Molli, and A. Imine. Proving correctness of transformation functions in collaborative editing systems. Technical Report 5795, INRIA, Dec. 2005.

AUTHORS PROFILE

Frist Author Santosh Kumawat

Mtech Scholar
Poornima College of Engineering
Rajasthan Technical University(RTU)
34, Meera Kutir, Jain Mohalla,
Main Bazar, Sanganer, Jaipur, Rajasthan, India-302029

Second Author Ajay Khunteta
Asst Prof. Dept. of CS
Poornima College of Engineering
Jaipur, Rajasthan, India