Kumawat and Khunteta *Technol Innov Educ (2015) 1:2*
DOI 10.1186/s40660-015-0003-z

Technology, Innovation and Education
a SpringerOpen Journal

SHORT REPORT Open Access

Cut-paste string operation for collaborative groupware applications

Santosh Kumawat[1*] and Ajay Khunteta[2]

*Correspondence: santoshkumawat82@ymail.com
[1] School of Engineering and Technology, Poornima University, IS-2027 To 2031 Ramchandrapura P.O. Vidhani Vatika Sitapura Extension, Jaipur 303905, Rajasthan, India
Full list of author information is available at the end of the article

Abstract

Operational transformation (OT) is a concurrency control approach in multi-user collaborative groupware system. With OT, local operations do not delay and shared document could be edited by users at any time. Most of OT algorithms cannot be formally proved. Till date only two character-wise primitive operations Insert and Delete are proposed and these operations have time complexity of the order of $O(|H|^2)$ where H is the history buffer of operations. In this paper, a new algorithm for cut-paste string operation is proposed with time complexity and space complexity of the order of $O(|H|)$. It reduces the time and space consumption in performing composite string operation—Cut-Paste. The proposed algorithm also handles overlapping and splitting of operations. A modified algorithm GSMRITFDD is also proposed that has removed all faults of existing ITDD algorithm.

Keywords: Operational Transformation Algorithms, Concurrency control, Distributed systems, Collaborative system, Groupware system

Background

Groupware system is a multiple user system in which the operations of each user must quickly be propagated to all other shared users (e.g., multi-player game, real-time computer conferencing).

Groupware system requires sharing of data, fine granularity, concurrency control and fast response times. Consistency and high local responsiveness are specific requirements for multi-user systems (Ellis and Gibbs 1989; Sun et al. 1998). Concurrency control protocols are used to remove inconsistency in the multi-user transactions systems such as Relational Databases, Distributed Systems and Groupware Systems.

Theorem 1 *In a consistent shared environment which has replicated data after execution of all operations all have same data.*

Traditional concurrency control methods such as Locking and Reversible Execution may cause the loss of some user interaction results. These methods are not suitable for distributed interactive applications which require fast local response satisfying user intentions, intention consistency and convergence. In past 15 years, OT is an acceptable method for consistency and concurrency maintenance in multi-user group editors such

as group editors and Google Wave1 (Davis et al. 2002; Ellis and Gibbs 1989; Sun et al. 1998a, b). When compared to other concurrency control methods, OT achieves convergence, causality and intention preservation without sacrificing local responsiveness and concurrent work (Sun et al. 1998a). OT allows users to edit any part of the data which is shared by other users (Bentley and Dourish 1995).

Review of OT algorithms

In last 25 years of OT algorithms, there are two main challenges in OT: First, most of algorithms require an informal condition known as "Intention Preservation" and their correctness cannot be formally proved. Second: except for (Ressel et al. 1996), all existing OT algorithms only support two character-based operations called insert and delete.

The proposed work studied major OT algorithms (Table 1) including Distributed Operation Transformation (dOPT) algorithm (Ellis and 1989), the Generic Operational Transformation (GOT) algorithm (Sun et al. 1998), GOT Optimized (GOTO) algorithm (Sun et al. 1998), State Difference Transformation (SDT) algorithm (Li and Li 2007), SCOT2 (Suleiman et al. 1998), SCOT 3/4 algorithm (Vidot et al. 2000), Adopted (adOPTed) algorithm (Ressel et al. 1996), Admissibility-Based Transformation (ABT) algorithm (Li and Li 2010), ABT-Undo (ABTU) Algorithm (Li and Li 2007), admissibility-based sequence transformation (ABST) (Shao et al. 2010a, b), and Admissibility-Based Transformation with Strings (ABTS) algorithm (Shao et al. 2009).

The ABT (Li and Li 2007) introduces a correctness criterion called admissibility preservation, in which correctness of admissibility-based transformation (ABT) functions has formally proved. ABTU arranges the operation history in total effects-relation order and improves the time complexity to O(IHI). In the available literature, only GOT, GOTO and ABTS algorithms support string-wise operations. The time complexity of GOT and GOTO algorithms are $O(|H^2|)$ but time complexity of ABTS is only O(|H|). So due to less time complexity, ABTS is better string-based OT algorithm as compared to GOT and GOTO. The proposed work is focused on string-based OT algorithm which is based on ABT framework.

OT framework

Most of existing OT algorithms have developed under a well-accepted framework with a condition that algorithm cannot be formally proved. In addition, they generally support two character-based primitive operations like insert and delete in a linear data structure. Only three algorithms called as GOT, GOTO and ABTS support primitive string operations. This research paper proposes a new novel OT algorithm for composite string operation Cut-Paste with existing primitive operations.

The insert(p, s) and delete(p, s), insert and delete a string 's' at position 'p' in the shared data, respectively. The proposed work has introduced a new composite string operation Cut-Paste (p1, s, p2) which cut from position 'p1' and paste it at position 'p2' in shared data with less time complexity.

In OT, shared data are like a linear string 's' of atomic characters in which objects are referred by their positions 'p' starting from zero in the string and consider two only primitive string operations, called, insert(p, s) and delete(p, s). At site, there exists a common definition state 's' for all operations. The standard notations are summarized in Table 2 (Shao et al. 2009).

Table 1 Comparison of OT Algorithms

Table	DOPT	AdOPTed	GOT	GOTO	SOCT2	SOCT3	SOCT4	SDT	ABT	ABTS
Time complexity	Consumes more time	Less than dOPT	A bit less than adOPTed	O(I H^2 I) which is slower than ABT	O(I H^2 I) which is slower than ABT	A bit more than SCOT4	A bit more than ABT	O(I H^2 I) which is slower than	O(I H^2 I)	O(|H|)
Support 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|)

Table 2 Standard Notations

Notations	Description
o.id	Id of site that generates operation o
o.type	Type of operation o, i.e., either insert or delete
o.pos	Position of operation o
o.str	String insert/delete by o
o1→o2	o1 occurs before o2
o1||o2	o1 and o2 are concurrent
o1Uo2	o1 and o2 are contextually equivalent
o1→o2	o1 and o2 are contextually serialized
[o1,o2]	An ordered list of two operations o1 and o2
<o1,o2>	Two operations in sequence
|L|	Number of objects in list L
L1.L2	Concatenation of two lists L1 and L2
s[i:len]	Substring of string s starts from position i of length len
Sq	A sequence is a special list
R1 = [o1.start, o1.end]	Operation region of operation o1 is R1 which starts from o1.start & ends at o1.end
cut-paste(p1, s, p2)	Cut a string s from p1 and paste at p2
exec(oi)	Execution of operation oi

Methods

The proposed algorithm for cut-paste string composite operation ITCutPaste (Inclusive Transformation of Cut-Paste) is based on inclusive transformation of operational transformation. The algorithm ABTS (Shao et al. 2009), already contains ITII (Inclusive Transformation of Insert–Insert)/ITID (Inclusive Transformation of Insert–Delete)/ITDD (Inclusive Transformation of Delete–Delete)/ITDI (Inclusive Transformation of Delete–Insert) algorithms for transformation of insert and delete string operations. So to perform cut-paste operation, first the substring 's' get deleted from p by operation delete(s, p) and then 's' get inserted at position q by operation insert(s, q). Here, delete(s, p) and insert(s, q) are independent operations. At all sharing sites, transformation functions corresponding to both delete(p, s) and insert(q, s) get called which results in increase in the time complexity.

This research work proposes a single transformation algorithm for composite string operation cut-paste. So all sites require only single transformation function for cut-paste operation. The time complexity of transformation function ITCutPaste for cut-paste operation is same as transformation function for insert or delete ITDI/ITID. It is based on inclusive transformation and ABT Framework so it can be formally proved.

Type of operations

ITCutPaste consists of operations o1, o2 as input and o1′ as output. Operations o1, o2 and o1′ may be string or character operations which operate on shared data in multi-user groupware collaborative applications like group editors.

Definition 1 o1 and o2 are contextually equivalent o1||o2, o1Uo2 and if input is o1 and output then output should be o2→o1′.

Design and analysis of algorithms

ITCutPaste consists many sub-algorithms ITDCp (Inclusive Transformation of Deletion with Cut-Paste), ITICp (Inclusive Transformation of Insertion with Cut-Paste), ITCpI (Inclusive Transformation of Cut-Paste with Insertion), and ITCpD (Inclusive Transformation of Cut-Paste with Deletion). The proposed algorithm satisfies causality and admissibility preservation.

Definition 2 If we have exec(o_i), then all exec(o_{i-1}) must be completed then only oi satisfies causality.

Definition 3 If o1Uo2 then IT(o1,o2) satisfies admissibility. It does not have inconsistent order at shared environment.

Theorem 2 Let H is admissible history of operation which satisfies causality. Also, sq is a sequence of operations and sq~H. Then, if o is executed in state s, we have exec(s, H). There should be sq′~ (sq.o) and sq′ must be consistent.

The ITCutPaste(o1, o2) transforms o1 with another operation o2 with output of this function is o1′. The precondition of ITCutPaste(o1, o2) is o1Uo2 and the post-condition is o2→o1′. The following "Type of operations" presented the algorithm ITDCp and "Design and analysis of algorithms" presented the algorithm ITICp. In algorithm ITCutPaste if o1 operation type is insert and o2 operation type is CutPaste then ITICp is called and if o1 operation type is delete and o2 operation type is CutPaste then ITDCp is called. If o1 operation type is CutPaste and o2 operation type is insert, then ITCpI is called and if o1 operation type is CutPaste and o2 operation type is delete then ITCpD is called.

Algorithm: ITCutPaste(o1, o2): o1′

```
{ o1′ ← o1;
if (o1.type = "Insert" and o2.type = "CutPaste")
{ ITICp(o1, o2): o1′;}
else if (o1.type = "Delete" and o2.type = "CutPaste")
{ ITDCp(o1, o2): o1′;}
else if (o1.type = "CutPaste" and o2.type = "Insert")
{ ITCpI(o1, o2): o1′;}
else if (o1.type = "CutPaste" and o2.type = "Delete")
{ ITCpD(o1, o2): o1′;}
endif
return o1′;
}
```

Algorithm ITDCp

Algorithm ITDCp takes as parameters o1 and o2 and return o1′. Here, o1 is deletion operation and o2 is Cut-Paste operation. Here, precondition is o1Uo2 and post-condition is o2→o1′. GSMRITFDD(o1, o2) used in Algorithm ITDCp is transformation algorithm to transform o1 (delete) and o2 (delete) operations.

Algorithm: ITDCp(o1, o2): o1′

```
{
o1′ ← o1;
if((o1.pos+|o1.str|) < o2.pos or o1.pos >= p)
{o1′ ← o1;}
else if(o1.pos  > ( o2.pos+|o2.str|) && o1.pos < p)
{o1′.pos ← o1.pos - |o2.str|;}
else
{o1′ ← GSMRITFDD(o1, o2);}
}
  endif
  return o1′;

}
```

Algorithm ITICp

Algorithm ITICp takes as parameters o1 and o2 and return o1′. Here, o1 is insertion operation and o2 is Cut-Paste operation. Here, precondition is o1Uo2 and post-condition is o2→o1′.

Algorithm: ITICp (o1, o2): o1′

```
{

o1′ ← o1;
if(o1.pos <= o2.pos && o1 >= p)
{o1′ ← o1;}
elseif(o1 >= o2.pos && o1 <= o2.pos + |o2.str|)
{o1′.pos = o2.pos;}
elseif(o1.pos > (o2.pos + |o2.str| && o1 < p)
{ o1′.pos = o1.pos - |o2.str| ;}
}
```

Algorithm GSMRITFDD

The new algorithm GSMRITFDD is proposed that removed all faults of existing ITDD (Shao et al. 2009) for inclusive transformation of two deletions and work well in all possible cases. It needs to consider the following cases regarding the relations between the two target regions, R1 = s [o1.pos :(|o1.str| + o1.pos)] and R2 = s[o2.pos:(|o2. pos| + |o2.str|)]

Algorithm GSMRITFDD (o1, o2):o1′

```
1: o1′←o1
2: o1_Length←|o1.str| and o2_Length←|o2.str|
3: o1_start← o1.pos and
4: o1_end← o1.pos+|o1.str|
5: o2_start← o2.pos
6: o2_end← o2.pos+|o2.str|
7: if o2_end<= o1_start then
8: o1′.pos← o1_start- o2_Length
9: else if o2_start>= o1_end then
10: return o1′
11: else if o1_start < o2_start and o1_end <= o2_end then
12: o1′.str←o1.str[0: o2_start - o1_start]
13: else if o1_start >= o2_start and o1_end > o2_end then
14:o1′.pos← o2_start
15: o1′.str←o1.str[(o2_end - o1_start): ]
16: else if o1_start >= o2_start and o1_end <= o2_end then
17: return null
18: else if o1_start < o2_start and o2_end < o1_end then
19: oLpart←oRpart← o1
20: oLpart.str← o1.str[0: (o2_start - o1_start)]
21: oRpart.pos← o2_start
22: oRpart.str← o1.str[(o2_end - o1_start ): ]
23: o1′.sol←[ oLpart, oRpart]
24: endif
25: return o1′
```

Consider the following cases regarding the relations between the two target regions, R2 = s [o2.pos: (o2.pos + lo2.strl)] and R1 = s [o1.pos: (lo1.strl + o1.pos)].

1. (By line-9) When R2 is completely on the right of R1. In this case, Deletion of R2 does not affect o1 and hence o1 is returned as-is.
2. (By line-7) If R1 is on the right of R2. In this case, after R2 is deleted, we shift o1′.pos by 1o2.strl characters to the left to get transformed string operation o1′.
3. (By line-16) R1 is included in R2. In this case, after o2 is executed, R1 is already deleted. So there is no longer need to execute o1. That is why it returns an empty operation ¢.
4. (By line-13) When R2 partially overlaps with R1 around the left border of R1. After o2 is executed, the left part of R1 is already deleted. Hence, in this case, we need to reset o1.pos so that it will start from (o2.pos). So o1.str only needs to include the right part that is not deleted by o2, starting from (o2.pos + lo2.strl)—o1.pos in the original o1.str.
5. (By line-11) When R2 partially overlaps with R1 around the right border of R1. In this case, this is similar to case (4). After o2 is executed, o1 only needs to delete the left part that is not deleted by o2.
6. (By line-18) R2 is included in R1. The deletion of R2 within R1 divides R1 into three parts, among which the middle overlapping part is already deleted by o2. Hence, o1

must be split into two sub-operations that delete the two remaining substrings left and right, respectively.

Result and discussion

We have implemented ABTS and ITCutPaste in lab using Qualnet and ASP.Net software in multi-user environment. The Benchmark Dataset of online Group Editor is used to verify it. We have implemented existing IT algorithm ABTS and our proposed algorithm ITCutPaste to perform string composite operation cut-paste. We experimentally conclude that ABTS consumed more time by a factor in multiple of |H| as compared to ITCutPaste. ITCutPaste satisfies causality preservation and admissibility preservation.

Correctness proof

If we use ABTS to transform cut-paste string operation and if o1 = "insert" and o2 = "CutPaste", then we need ITID and ITII both for it. ITID applies for all o2.pos ≤ (o1.pos + |o1.str|) and ITII for all o2.pos ≤ o1.pos. Now, ITICp apply only for (o1.pos > o2.pos && o1.pos < p). Time complexity of ITII, ITID and ITICp is of same order. Since ITICp is for less number of operations in less range, so it consumes less or equal time as compared to ITID/ITII. ITICp could not consume more time as compared to ABTS(ITID + ITII) to operate same cut-paste string operation.

If we use ABTS to transform cut-paste string operation and if o1 = "delete" and o2 = "CutPaste", then we need ITDI and ITDD both for it. ITDI applies for all o2.pos < (o1.pos + |o1.str|) and ITDD for all o1.pos >= (o2.pos + |o2.str|). Now, ITDCp apply only for (o1.pos > o2.pos && o1.pos < p). Time complexity of ITDD, ITDI and ITDCp is of same order. Since ITDCp is for less number of operations in less range, so it consumes less or equal time as compared to ITDI/ITDD. ITDCp could not consume more time as compared to ABTS (ITDI +ITDD) to operate same cut-paste string operation.

Similarly, time complexity of ITCpD is of same order of ITDD/ITID. But to transform o2 = "delete" and o1 = "CutPaste" in case of ABTS, both ITDD and ITID need to get called for greater range of operations but for ITCutPaste only ITCpD is sufficient for same or less range of operations as compared to either ITDD or ITID of ABTS. Also, time complexity of ITCpI is of same order of ITDI/ITII. But to transform o2 = "insert" and o1 = "CutPaste" in case of ABTS, both ITDI and ITII need to get called but for ITCutPaste only ITCpI is sufficient for same or less range of operations as compared to either ITDI or ITII of ABTS. So ITCutPaste is more efficient.

ABTS time consumption for string operation cut-paste is if 2 O(|H|) then ITCutPaste consumes only O(|H|) time and space complexity.

A graphical representation of ABTS and ITCutPaste time complexity for various cases is shown below.

Conclusion

OT is the best method for concurrency and consistency control in multi-user groupware systems. Most of OT algorithms support character operations and very few support string primitive operations like insert and delete. The proposed algorithm ITCutPaste for string composite operation cut-paste works well in all conditions and handles

overlapping of operations. It is based on ABT framework and can be formally proved. It has time complexity and space complexity O(|H|) (where H is history buffer) similar to ABTS. The proposed algorithm GSMRITFDD has removed all faults of existing ITDD for inclusive transformation of two deletions and works well in all possible cases.

Authors' contributions

SK made substantial contributions to conception and design, acquisition of data, and analysis and interpretation of data; has been involved in drafting the manuscript or revising it critically for important intellectual content; has given final approval of the version to be published, and agrees to be accountable for all aspects of the work in ensuring that questions related to the accuracy or integrity of any part of the work are appropriately investigated and resolved. AK provided full guidance and support in all of the above works. Both authors read and approved the final manuscript.

Author details

[1] School of Engineering and Technology, Poornima University, IS-2027 To 2031 Ramchandrapura P.O. Vidhani Vatika Sitapura Extension, Jaipur 303905, Rajasthan, India. [2] Department of Computer Science and Engineering, Poornima College of Engineering, Jaipur, India.

Competing interests

The authors declare that they have no competing interests.

Received: 6 July 2015 Accepted: 10 October 2015
Published online: 23 November 2015

References

Bentley R, Dourish P (1995) Medium versus mechanism: Supporting collaboration through customization. In: ECSCW'95 Proceedings

Davis AH, Sun C, Lu J (2002) Generalizing operational transformation to the standard general markup language. In: Proceedings of ACM Conference Computer-Supported Cooperative Work (CSCW'02), pp 58–67

Ellis CA, Gibbs SJ (1989) Concurrency control in groupware systems. In: ACM SIGMOD'89 Proceedings, Portland Oregon, pp 399–407

Li R, Li D (2007) A new operational transformation framework for real-time group editors. IEEE Trans Parallel Distrib Syst 18(3):307–319

Li D, Li R (2010) An admissibility-based operational transformation framework for collaborative editing systems. In: Computer Supported Cooperative Work (2010) 19:1–43, Springer 2009. doi:10.1007/s10606-009-9103-1

Ressel M, Nitsche-Ruhland D, Gunzenha¨user R (1996) An integrating, transformation-oriented approach to concurrency control and undo in group editors. In: Proceedings ACM Conference on Computer-Supported Cooperative Work (CSCW'96), pp 288–297

Shao B, Li D, Gu N (2009) ABTS: a transformation-based consistency control algorithm for wide-area collaborative applications. In: 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing. Collaborate Com 2009. Vol, 1, No. 10, pp 11–14. doi:10.4108/ICSTCOLLABORATECOM2009.8271

Shao B, Li D, Gu N (2010a) An algorithm for selective undo of any operation in collaborative applications. In: ACM

Shao B, Li D, Gu N (2010b) A Fast operational transformation algorithm for mobile and asynchronous collaboration. IEEE Trans Parallel Distrib Syst 21(12)

Suleiman M, Cart M, Ferrié J (1998) Concurrent operations in a distributed and mobile collaborative environment. In: Proceedings of the Fourteenth International Conference on Data Engineering, February. pp 23–27

Sun C, Ellis C (1998) Operational transformation in real-time group editors: issues, algorithms, and achievements. In: ACM CSCW'98, pp 59–68

Sun C, Jia X, Zhang Y, Yang Y, Chen D (1998) Achieving convergence, causality-preservation, and intention preservation in real-time cooperative editing systems. ACM Trans Comp Human Interact 5(1):63–108

Vidot N, Cart M, Ferrie J, Suleiman M (2000) Copies convergence in a distributed real-time collaborative environment. In: Proceedings of the 2000 ACM conference on Computer supported cooperative work. ACM Press New York, NY, USA. pp 171–180