Real Differences between OT and CRDT under a General Transformation Framework for Consistency Maintenance in Co-Editors

Chengzheng Sun, Nanyang Technological University, Singapore
David Sun, Codox Inc., United States
Agustina, Nanyang Technological University, Singapore
Weiwei Cai, Nanyang Technological University, Singapore

OT (Operational Transformation) was invented for supporting real-time co-editors in the late 1980s and has evolved to become a core technique used in today’s working co-editors and adopted in major industrial products. CRDT (Commutative Replicated Data Type) for co-editors was first proposed around 2006, under the name of WOOT (WithOut Operational Transformation). Follow-up CRDT variations are commonly labeled as “post-OT” techniques and have made broad claims of superiority over OT solutions, in terms of correctness, time and space complexity, etc. Over one decade later, however, OT remains the choice for building the vast majority of co-editors, whereas CRDT is rarely found in working co-editors. Contradictions between the reality and CRDT’s purported advantages have been the source of much confusion and debate in co-editing research and developer communities. What is CRDT really to co-editing? What are the real differences between OT and CRDT for co-editors? What are the key factors that may have affected the adoption of and choice between OT and CRDT for co-editors in the real world? A thorough examination of these questions is relevant not only to researchers who are exploring the frontiers of co-editing technologies and systems, but also to practitioners who are seeking viable techniques to build real world collaboration applications. Based on comprehensive reviews and comparisons on representative OT and CRDT solutions and working co-editors based on them, we present our main research discoveries, in relation to these questions and beyond, in a series of three independent and complementary articles. Also, we reveal facts and present evidences that refute CRDT claimed advantages over OT.

In this article, we present a general transformation framework for consistency maintenance in co-editors, which can be used to describe and study a range of consistency maintenance approaches, including OT and CRDT. Our study has made important and surprising discoveries, including that CRDT is like OT in following the general transformation approach, and CRDT is not natively commutative for concurrent operations in co-editors. Uncovering the hidden transformation nature and demystifying the commutativity property of CRDT provides much-needed clarity about what CRDT really is and is not to co-editing, which in turn helps bring out the real differences between OT and CRDT in correctness, complexity, system implementation and application. We hope the discoveries from this work help clear up common misconceptions and confusions surrounding OT and CRDT, and accelerate progress in co-editing technology for real world applications.

CCS Concepts: • Information Systems → Group and Organization Interfaces; Synchronous Interaction, Theory and Model.

KEYWORDS
Operational Transformation (OT), Commutative Replicated Data Type (CRDT), concurrency control, consistency maintenance, real-time collaborative editing, distributed/Internet/cloud computing technologies and systems, Computer Supported Cooperative Work (CSCW) and social computing.

1 INTRODUCTION

Real-time co-editors allow multiple geographically dispersed people to edit shared documents at the same time and see each other’s updates instantly [1,6,13,14,15,16,38,43,54,55,60,72,78]. One major challenge in building such systems is consistency maintenance of documents in the face of concurrent editing, under high communication latency environments like the Internet, and without imposing interaction restrictions on human users [13,54,55].

©2019 Copyright held by the authors. Submitted to arXiv.org on May 2, 2019.
Corresponding author: Chengzheng Sun, School of Computer Science and Technology, Nanyang Technological University, Singapore. Email: CZSun@ntu.edu.sg; URL: https://www.ntu.edu.sg/home/CZSun
Operational Transformation (OT) was invented to address this challenge [13,54,61,72] in the late 1980s. OT introduced a framework of transformation algorithms and functions to ensure consistency in the presence of concurrent user activities. The OT framework is grounded in established distributed computing theories and concepts, principally in concurrency and context theories [24,54,66,67,83,84]. Since its inception, the scope of OT research has evolved from the initial focus on consistency maintenance (or concurrency control) to include a range of key collaboration-enabling capabilities, including group undo [38,44,57,58,66,67], and workspace awareness [1,19,60]. In the past decade, a main impetus to OT research has been to move beyond plain-text co-editing [6,13,20,38,43,54,55,58,61,70,71,77], and support rich-text co-editing in word processors [60,65,68,82], HTML/XML Web document co-editing [10], spreadsheet co-editing [69], 3D model co-editing in digital media design tools [1,2], and file synchronization in cloud storage systems [3]. Recent years have seen OT being widely adopted in industry products as the core technique for consistency maintenance, ranging from battle-tested online collaborative rich-text editors like Google Docs1[11], to emerging start-up products, such as Codox Apps2.

A variety of alternative techniques for consistency maintenance in co-editors had also been explored in the past decades [14,16,18,41,72]. One notable class of techniques is CRDT3 (Commutative Replicated Data Type) for co-editors [4,5,8,25,32,37,39,40,41,45,47,48,79,80,81]. The first CRDT solution for plain-text co-editing appeared around 2006 [40,41], under the name of WOOT (WithOut Operational Transformation). One motivation behind WOOT was to solve the FT (False Tie) puzzle in OT [53,55] (also discussed in detail in [73]), using a radically different approach from OT. Since then, numerous WOOT revisions (e.g. WOOTO [80], WOOTH [4]) and alternative CRDT solutions (e.g. RGA [45], Logoot [79,81], LogootSplit [5]) have appeared in literature. CRDT has often been labeled as a "post-OT" technique that makes concurrent operations natively commutative, and does the job "without operational transformation" [40,41], and even "without concurrency control" [25]. CRDT solutions have made broad claims of superiority over OT solutions, in terms of correctness, time-space complexity, simplicity, etc. After over one decade, however, CRDT solutions are rarely found in working co-editors or industry co-editing products, and OT solutions remain the choice for building the vast majority of co-editors.

The contradictions between realities and CRDT’s purported advantages have been the source of much confusion and debate in co-editing research and developer communities4. What is CRDT really to co-editing? What are the real differences between OT and CRDT for co-editors? What are the key factors that may have affected the adoption of and choice between OT and CRDT for co-editors in the real world? We believe that a thorough examination of these questions is relevant not only to researchers who are exploring the frontiers of collaboration-enabling technologies and systems, but also to practitioners who are seeking viable techniques to build real world collaboration tools and applications.

To seek answers to these questions and beyond, we set out to conduct a comprehensive review and comparative study on representative OT and CRDT solutions and working co-editors based on

---

1 https://www.google.com/docs/about/
2 https://www.codox.io
3 In literature, CRDT can refer to a number of different data types [48]. In this paper, we focus exclusively on CRDT solutions for text co-editors, which we abbreviate as “CRDT” in the rest of the paper, though occasionally we use “CRDT for co-editors” for emphasizing this point and avoiding misinterpretation.
4 We posted an early version of our report on this work at https://arxiv.org/abs/1810.02137, on Octo-4, 2018, which attracted wide interests and discussions in public blogs (among both academics and practitioners) and private communications (between readers and authors). This link, at https://news.ycombinator.com/item?id=18191867, hosts some representative comments and opinions on various issues addressed in our prior article. For example, a well-known CRDT researcher commented: “The argument of Sun’s paper seems to be that CRDTs have hidden performance costs. Perhaps this is true. This completely misses the main point. OT is complex, the theory is weak, and most OT algorithms have been proven incorrect (…). AFAIK, the only OT algorithm proved correct is TTF, which is actually a CRDT in disguise. In contrast, the logic of CRDTs is simple and obvious. We know exactly why CRDTs converge. … Disclaimer: I did not read the paper in detail, just skimmed over it.” The above comments basically reiterated and further amplified some typical CRDT claims and criticisms against OT, which reconfirms the liveness of these issues, and warrants thorough examination of them. In the prior article and the extended series, we provide facts and evidences that refute all these CRDT claims. Readers may check whether any of the above mentioned points missed in our articles and make independent judgement on these issues.
them, which are available in publications or from publicly accessible open-source project repositories. In this work, we explored what, how, and why OT and CRDT solutions are different and the consequences of their differences from both an algorithmic angle and a system perspective. From this exploration, we made a number of discoveries, some of which are rather surprising. One such discovery is that CRDT is a transformation approach to consistency maintenance and is not natively commutative to concurrent operations in co-editors. We have also examined major CRDT claims over OT, and provided evidences that refute those claims.

In this work, we focus on OT and CRDT solutions to consistency maintenance in real-time co-editing, as it is the foundation for other co-editing capabilities, like group undo and issues related to non-real-time co-editing, which we plan to cover in future work. We know of no existing work that has made similar attempts.

The topics and bulk of outcomes from this study are comprehensive, complex and diverse, which are of interest and accessible to readers with different needs and backgrounds. To cope with the complexity and diversity of topics and readership and take into account of feedback to a prior version of our report on this work (see footnote 4), we have organized the outcome materials into three parts and presented them in a series of three independent and complementary articles.

1. **Real differences between OT and CRDT under a general transformation framework for consistency maintenance in co-editors** (this article). We review the basic ideas of OT and CRDT and present a general transformation framework for consistency maintenance in co-editors. Furthermore, we reveal that CRDT is like OT in following the general transformation approach, and is not natively commutative for concurrent operations in co-editors. Uncovering the hidden transformation nature and demystifying the commutativity property of CRDT provides much-needed clarity about what CRDT really is and is not to co-editing, which in turn brings out the real differences between OT and CRDT for co-editors. Materials in this article are presented at high levels and require no in-depth co-editing technical background from readers.

2. **Real differences between OT and CRDT in correctness and complexity for consistency maintenance in co-editors** [73]. We dissect and examine representative OT and CRDT solutions, and explore how different basic approaches – the concurrency-centric approach taken by OT versus the content-centric approach taken by CRDT – had resulted in different technical challenges and consequential correctness and complexity issues. Moreover, we reveal hidden algorithmic flaws with some representative CRDT solutions, and discuss common myths and facts related to correctness, time and space complexity, and simplicity of OT and CRDT solutions. Materials in Part II are technical in nature, but a large part of them are described at high levels and should be understandable by people with general knowledge of OT and CRDT literature. However, in-depth understanding of the technical contents in this article require advanced co-editing technical background from readers.

3. **Real differences between OT and CRDT in building co-editing systems and real world applications** [74]. We examine the role of building co-editing systems in OT and CRDT research, and the consequential differences in the adoption and choice between OT and CRDT in real world co-editors. In particular, we review the evolution of co-editors from research vehicles to real world applications, and discuss representative OT-based co-editors and alternative approaches in industry products and open source projects. Moreover, we evaluate CRDT-based co-editors in relation to published CRDT solutions, and clarify myths surrounding system implementation and “peer-to-peer” co-editing. Materials in this article should be understandable by people with general knowledge in co-editing, and of particular interest to practitioners seeking viable techniques for building real world applications.

For improving readability and self-containment, we include the same introduction and references in three articles. At the end of each article, the conclusion section summarizes main results and contributions covered in that individual article. In the third article [74], the conclusion gives a summary of major results and contributions presented in the series of three articles.
2 BASIC IDEAS OF A GENERAL TRANSFORMATION APPROACH

Modern real-time co-editors have commonly adopted a replicated architecture: the editor application and shared documents are replicated at all co-editing sites. A user may directly edit the local document replica and can see the local effect immediately; local edits are promptly propagated to remote sites for real-time replay there. There are two basic ways to propagate local edits: one is to propagate the edits as operations [13,41,54,55,79]; the other is to propagate the edits as states [14]. Most real-time co-editors, including those based on OT and CRDT, have adopted the operation approach for propagation for communication efficiency, among others. The operation approach is assumed for all editors discussed in the rest of this article.

The central issue shared by all co-editors is: how an operation generated from one replica can be replayed at other replicas, in the face of concurrent operations, to achieve consistent results across all replicas. Co-editors are generally required to meet three consistency requirements [55]: the first is causality-preservation, i.e. operations must be executed in their causal-effect orders, as defined by the happen-before relation [24]; the second is convergence, i.e. replicas must be the same after executing the same collection of operations; and the third is intention-preservation, i.e. the effect of an operation on the local replica from which this operation was originally generated must be preserved at all remote replicas in the face of concurrency.

A general approach to achieving both convergence and intention-preservation, invented in co-editing research, is based on the notion of transformation, i.e. an original operation is transformed (one way and another) into a new version, according to the impact of concurrent operations, so that executing the new version on a remote replica can achieve the same effects as executing the original operation on its local replica [55]. This approach allows concurrent operations to be executed in different orders (i.e. being commutative) but in non-original forms5. Causality-preservation can be achieved by adopting numerous suitable distributed computing techniques [13,24,54], without involving the aforementioned transformation.

The transformation approach can be illustrated by using a real-time plain text co-editing scenario in Fig. 1-(a). The initial document state "abe" is replicated at two sites. Under the transformation-based consistency maintenance scheme, users may freely edit replicated document states to generate operations. Two operations, $O_1 = D(1)$ (to delete the character at position 1) and $O_2 = I(2, "c")$ (to insert character c at position 2), are generated by User A and User B, respectively. These two operations are concurrent with each other as they are generated without the knowledge of each other [24,54]. The two operations are executed as-is immediately at local sites to produce "ae" and "abe", respectively; and then propagated to remote sites for replay.

In the absence of any consistency maintenance scheme, the two operations would be executed in their original forms and in different orders, due to network communication latency, at the two sites, which would result in inconsistent states "aec" (under the shadowed cross at User A) and "ace" (at User B), as shown in Fig. 1-(a). Under the transformation-based consistency maintenance, however, a co-editor may execute a remote operation in a transformed form that takes into account the impact of concurrent operations, or concurrency-impact in short. In this example:

- At User A, $O_1$ has left-shifting concurrency-impact on $O_2$. So, the transformation scheme creates a new $O_2' = I(1,"c")$ from the original $O_2 = I(2, "c")$, to insert "c" at position 1.
- At User B, $O_2$ has no shifting concurrency-impact on $O_1$. So, the original $O_1 = O_1' = D(1)$ can be applied to delete "b" at position 1.

Executing $O_2'$ at User A and $O_1'$ at User B, respectively, would result in the same document state "ace", which is not only convergent, but also preserves the original effects of both $O_1$ and $O_2$, thus meeting the intention-preservation requirement [54,55]. We draw attention to the fact that, as seen in Fig. 1-(a), $O_1$ and $O_2$ are executed in different orders at two sites but achieve the same result, which illustrates that the transformation approach has the capability of making concurrent operations commutative among replicated documents.

---

5 In contrast, an alternative approach, called serialization, forces all operations to be executed in the same order and in original forms [13,16,54]. It has been shown the serialization approach is unable to achieve intention-preservation [55].
The consistency maintenance problem and solution illustrated in Fig. 1-(a) should look familiar to readers with some background in OT. Indeed it has often been used to explain basic OT ideas [13,54,55,70,71]. What might be surprising to many is that the same formulation of problem and solution applies equally to CRDT as well: CRDT was proposed to address the same consistency maintenance issues in co-editors, and has actually followed the same transformation approach.

OT has been known for its very capability of making concurrent operations commutative among replicated documents long before CRDT appeared. What has been mysterious and confusing to many is the notion that CRDT achieves commutativity of concurrent operations natively or by design, whereas OT achieves commutativity after the fact [47,48]. In this work (particularly Sections 4 and 5), we demystify the CRDT native commutativity and reveal CRDT has to achieve the same OT commutativity after the fact as well.

Revealing the basic ideas of the general transformation approach shared by both OT and CRDT at the start (with more elaborations in Section 3) helps set a common ground for examining the real differences between OT and CRDT: their radically different approaches to realizing the general transformation, and the consequential correctness and complexity issues associated with each approach, which are discussed in this and other articles in this series [73,74].

3 DIFFERENT APPROACHES TO REALIZING TRANSFORMATION

In the next two subsections, we present the basic elements of OT and CRDT for co-editors, and use the same co-editing scenario in Fig. 1-(a) to illustrate how OT and CRDT realize the general transformation. Rather than a purely algorithmic discussion, we take a user-oriented end-to-end perspective, i.e. from the point when an operation is generated from a local editor by a user, all the way to the point when this operation is replayed in a remote editor seen by another user. We give step by step illustrations of the general process of handling an operation at both local and remote sites under both approaches, so that the subtle but key differences between OT and CRDT can be contrasted (the devil is in the details).
3.1 The OT Approach

3.1.1 Key Ideas and Components

An OT solution for consistency maintenance typically consists of two key components\(^6\): generic control algorithms for managing the transformation process; and application-specific transformation functions for performing the actual transformation (or manipulation) on concrete operations. At each collaborating site, OT control algorithms maintain an operation buffer for saving operations that have been executed and may be concurrent with future operations.

The life cycle of a user-generated operation in an OT-based co-editor can be sketched below.

- When an operation is generated by a user at a collaborating site, this operation is immediately executed on the local document state visible to the user. Then, this operation is timestamped to capture its concurrency relationship with other operations and saved in the local buffer. Next, the timestamped operation is propagated to remote sites via a communication network.
- When an operation arrives at a remote site, it is accepted according to the causality-based condition [13,24,54]. Then, control algorithms are invoked to select suitable concurrent operations from the buffer, and transformation functions are invoked to transform the remote operation against those concurrent operations to produce a transformed operation (a version of the remote operation is also saved in the buffer). Finally, the transformed operation is replayed on the document visible to the remote user.

For a plain-text co-editor with a pair of insert and delete operations, a total of four transformation functions, denoted as \(T_{ii}, T_{id}, T_{di},\) and \(T_{dd}\), are needed for four different operation type combinations [54,61,70,71]. Each function takes two operations, compares their positional relations (e.g. left, right, or overlapping) to derive their concurrency impacts on each other, and adjusts the parameters of the affected operation accordingly. When extending an OT solution to editors with different data and operation models, transformation functions need to be re-defined, but generic control algorithms need no change.

3.1.2 A Working Example for OT

In Fig. 1-(b), we illustrate how the key components of an OT solution work together to achieve the consistent result in Fig. 1-(a). Each co-editing site is initialized with the same external document state “abe", and an empty internal buffer BUF.

**Local Operations Handling.** User A interacts with the external state to generate \(O_1 = D(I)\), which results in a new state "ae". Internally, the OT solution at User A would do the following:

1. Timestamp \(O_1\) to produce an internal operation \(O_1(t)\).
2. Save \(O_1(t)\) in BUF = \([O_1(t)]\).
3. Propagate \(O_1(t)\) to the remote site.

Concurrently, User B interacts with the external state to generate \(O_2 = I(2,"c")\), which results in a new state "abce". Internally, the OT solution at User B would do the following:

1. Timestamp \(O_2\) to produce an internal operation \(O_2(t)\).
2. Save \(O_2(t)\) in BUF = \([O_2(t)]\).
3. Propagate \(O_2(t)\) to the remote site.

**Communication and Operation Propagation:** The basic OT approach described here is independent of specific communication structures or protocols (more elaboration on this point later in this article). What is noteworthy here is that under the OT approach, operations propagated among co-editing sites are position-based operations.

---

\(^6\) In this paper, we focus exclusively on OT solutions that separate generic control algorithms from application-specific transformation functions [1-3,6,10,11,13,15,20,26,31,34,36,38,43,44,49-58,60,65-72,76-78,82-84], as they represent the majority and mainstream OT solutions, on which existing OT-based co-editors are built. In literature, however, there are other OT solutions (e.g. [27-30,46]), in which control procedures are not generic but dependent on specific types of operation and data, and transformation procedures may examine concurrency relationships among other operations as well. In those OT solutions, "control procedure and transformation functions are not separated as in previous works – instead, they work synergistically in ensuring correctness" [30], and different correctness criteria were used as well [27-30,46,61].
Remote Operation Handling. When $O_2(t)$ arrives at User A, OT would do the following:
1. Accept $O_2(t)$ for processing under certain conditions (e.g. causal ordering [24]).
2. Transform $O_2(t)$ into $O_2'(t)$ by:
   a. invoking the control algorithm to get $O_1(t)$ from BUF, which is concurrent and defined on the same initial document state with $O_2(t)$; and
   b. invoking the transformation function $Tid(O_2, O_1)$ to produce a transformed operation $O_2' = I(1, \"c\")$. The $Tid$ function works by comparing the position parameters 2 and 1 in $O_2$ and $O_1$, respectively, and derives that $O_2$ is performed on the right of $O_1$ in the linear document state, and hence adjusts $O_2$ position from 2 to 1 to compensate the left shifting effect of $O_1$.
3. Save $O_2'(t)$ in BUF = [ $O_1(t)$, $O_2'(t)$ ].
4. Apply $O_2' = I(1, \"c\")$ on \"ae\" to produce \"ace\”.

When $O_1(t)$ arrives at User B, OT would do the following:
1. Accept $O_1(t)$ for processing under certain conditions (e.g. causal ordering [24]).
2. Transform $O_1(t)$ into $O_1'(t)$ by:
   a. invoking the control algorithm to get $O_2(t)$ from BUF, which is concurrent and defined on the same initial document state with $O_1(t)$; and
   b. invoking the transformation function $Tdi(O_1, O_2)$ to produce a new operation $O_1' = D(1)$, which happens to be the same as the original $O_1$ because the $Tdi$ function derives (based on the position relationship 1 < 2) that $O_1$ is performed on the left of $O_2$ in the linear state, hence its position is not affected by $O_2$.
3. Save $O_1'(t)$ in BUF = [ $O_2(t)$, $O_1'(t)$ ].
4. Apply $O_1' = D(1)$ on \"abce\" to delete \"b\"; the document state becomes: \"ace\”.

There is no need to store operations in the buffer indefinitely. As soon as there is no future operation that could possibly be concurrent with the operations in the buffer (a general garbage collection condition for OT) [55,67,84], those operations can be garbage collected and the buffer can be reset, i.e., BUF = [ ].

3.2 The CRDT Approach

3.2.1 Key Ideas and Components
WOOT [40,41] is the first CRDT solution [48] for consistency maintenance in co-editors. WOOT has two distinctive components. The first is a sequence of data objects, each of which is assigned with an immutable identifier and associated with either an existing character in the external document (visible to the user) or a deleted character (this internal object is then called a tombstone). The second is the identifier-based operations, which are defined and applicable on the internal object sequence only.

Notwithstanding the existence of a variety of CRDT solutions, the life cycle of a user-generated operation in all CRDT solutions is essentially the same, and can be generally sketched as follows.
• When a local operation is generated by a user, it is immediately executed on the document visible to the user; then this operation is given as the input to the underlying CRDT solution. The CRDT solution converts the external position-based input operation into an internal identifier-based operation, applies the identifier-based operation to the internal object sequence, and propagates the identifier-based operation, to remote sites via a suitable external communication service.
• When a remote identifier-based operation is received from the network, the CRDT solution accepts it according to certain execution conditions [24,41], applies the accepted operation to the internal object sequence, and converts the identifier-based remote operation to a position-based operation, which is finally replayed on the external document state visible to the user at a remote site.

It is worth pointing out that the above CRDT process of handling a user-generated operation (until replaying it at a remote site) naturally existed but was often obscured in published CRDT solutions.
For WOOT to work, an insert operation carries not only the identifier of the target object (i.e. the new character to be inserted), but also identifiers of two neighboring objects corresponding to characters that are visible to the user at the time when the insert was generated. The target identifier and neighboring object identifiers, together with tombstones in the object sequence, are crucial elements in WOOT's solution to concurrency issues related to the FT puzzle [40,41].

It should be pointed out that WOOT did not (and no other CRDT solution did) actually change the formats of the external document state or operations, which are determined by the editing application [9]. For consistency maintenance purpose, WOOT (and other CRDT solutions) created an additional object sequence as an internal state, identifier-based operations as internal operations, and special schemes that convert between external and internal operations, search target objects or locations, and apply identifier-based operations in the internal state (see more discussions on the nature of CRDT internal object sequences and operations in Sections 4 and 5).

3.2.2 A Working Example for CRDT
In Fig. 1-(c), we illustrate how the key functional components of WOOT work together to achieve the consistent result in a simple scenario in Fig. 1-(a). This example also serves as an illustration of the general CRDT process sketched above.

At the start, each co-editing site is initialized with the same document state "abe" (visible to the user), and the same internal state (IS) consisting of a sequence of objects corresponding to the initial external document state:

$$IS = @s <a,ida>@s,idb,v> <a,ida,ida,ide,v> <e,ide,idb, @e,v> <e>$$,

where $@$ and $@e$ are two special objects marking the start and end points of an internal state; each of other objects has five attributes, e.g. $<b, idb, ida, ide, v>$, where $b$ is the character represented by this object, $ida$ is the identifier for this object, $ida$ and $ide$ are the identifiers for the two neighboring objects, respectively, and $v$ indicates the character in this object is visible to the user (note: $iv$ indicates the character is invisible). An object identifier is made of two integers ($sid$, seq), where $sid$ is the identifier of the site that creates the object, seq is the sequence number of the operations generated at that site.

Local Operation Handling. User A interacts with the external document to generate a position-based operation $O_1 = D(L)$, which results in a new state "ae". WOOT handles $O_1$ as follows:

1. Convert the position-based $D(L)$ into the identifier-based $D(idb)$ by:
   a. searching the object sequence, with the index position 1 in $O_1$, to locate the target object $<b, idb, ida, ide, v>$ by counting only visible objects ($v = true$);
   b. creating an identifier-based $D(idb)$, where $idb$ is taken from $<b, idb, ida, ide, v>$.
2. Apply $D(idb)$ to the object sequence by setting $iv$ in the target object, which becomes a tombstone (also depicted by a line crossing the object in Fig. 1-(c)).
3. Propagate $D(idb)$, rather than $D(L)$, to User B.

Concurrently, User B interacts with the external document to generate a position-based operation $O_2 = I(2, "c")$, which results in a new state "abece". WOOT handles $O_2$ as follows:

1. Convert the position-based $I(2, "c")$ into the identifier-based $I(c,idc, idb, ide)$ by:
   a. searching the object sequence, with the index position 2 in $O_2$, to find the two visible neighboring objects between the insert position in the object sequence by counting visible objects;
   b. creating an identifier-based operation $I(c, idc, idb, ide)$, where $c$ is the character to be inserted, $ide$ is a new identifier for $c$, $idb$ and $ide$ are the identifiers of the two neighboring objects, respectively.
2. Apply $I(c, idc, idb, ide)$ into the object sequence by creating a new object $<c, ide, idb, ide, v>$ and injecting it at a proper location between the neighboring objects.
3. Propagate $I(c, idc, idb, ide)$, rather than $I(2, "c")$, to User A.

Communication and Operation Propagation: The basic CRDT approach is independent of specific communication structures or protocols (more elaboration on this point later in this article). What
is noteworthy here is that operations propagated under the CRDT approach are identifier-based operations, which is different from the OT approach.

**Remote Operation Handling.** At User B, the remote operation D(idb) is handled as follows:

1. Accept D(idb) for processing under certain conditions [24,41].
2. Apply D(idb) in the object sequence by:
   a. searching the object sequence, with the identifier idb in D(idb), to find the target <b, idb, ida, ide, v> with a matching identifier; and
   b. setting iv to the target object (to mark it as a tombstone).
3. Convert the identifier-based D(idb) into the position-based D(I), where the position parameter 1 is derived by counting the number of (visible) objects from the target object <b, idb, ida, ide, iv> to the start of the object sequence.
4. Apply D(I) on the external state to delete "b".

At User A, the remote operation I(c, idc, idb, ide) is handled as follows:

1. Accept I(c, idc, idb, ide) for processing under certain conditions [24,41].
2. Apply I(c, idc, idb, ide) in the object sequence by:
   a. searching the sequence, with identifiers idb and ide in I(c, idc, idb, ide), to find the two neighboring objects; and
   b. creating a new object <c, idc, idb, ide, v> and injecting it at a proper location between the two neighboring objects.
3. Convert the identifier-based operation I(c, idc, idb, ide) into a position-based operation I(I, "c"), where the position 1 is derived by counting the number of visible objects from the new object <c, idc, idb, ide, v> to the start of the object sequence.
4. Apply I(I, "c") on the external state.

Finally, both sites reach the same final external and internal states. In WOOT and its variations (WOOTO [80] and WOTH [4]), there exists no scheme to safely remove those tombstones. In some other tombstone-based CRDT solutions (e.g. RGA [45]), a garbage collection scheme was proposed to remove tombstones under certain conditions.

## 4 A GENERAL TRANSFORMATION FRAMEWORK

The concrete co-editing scenario (Fig. 1-(b) and (c)) is an instantiation of the general workflow of OT and CRDT solutions under a general transformation approach for text co-editors. This general transformation framework is distilled from a variety of OT, CRDT and other consistency maintenance solutions to co-editing, which is one main outcome from this work. We describe the main steps in the workflow, from local operation generation, handling and propagation, to remote operation acceptance, handling and replay, and the key actions in each step in Table 1.

### 4.1 Key Components

The general transformation framework includes the following key components:

1. The external state and operation models, which provide the working context of a transformation-based consistency maintenance solution:
   a. ES (External State), which represents the text sequence for text editors;
   b. EO (External Operation), which updates the text sequence, including insert(p, c) and delete(p), where p is a positional reference to the text sequence of ES, c is a character in ES (this parameter could be extended to a string of characters).
2. The external Communication and Propagation (CP) service, which is responsible for broadcasting operations among co-editing sites.
3. The core functional components of a transformation-based consistency maintenance solution:
   a. LOH (Local Operation Handler), which encapsulates the data structures and algorithms for handling local operations.
b. **ROH (Remote Operation Handler)**, which encapsulates the data structures and algorithms for handling remote operations.

As described in Table 1, both OT and CRDT take the same *position-based* input operation $EO_{in}$ (defined on $ES_{local}$) at the local site, and produce a transformed *position-based* output operation $EO_{out}$ (defined on $ES_{remote}$) at a remote site.

The modellings of the EO as *position-based* operations and the ES as a *sequence of characters* have been commonly adopted in OT and CRDT solutions. This data and operation modelling is neither accidental nor merely a modelling convenience, but is consistent with and well-supported by decades of practice of building text editors [9, 12, 23, 75]. We highlight that the use of position-based operations does not imply the text sequence must be implemented as an array of characters. The positional reference to the text sequence could be (and has commonly been) implemented in numerous data structures, such as an array of characters, the linked-list structures, the buffer-gap structures, and piece tables [9, 75].

Table 1 Describing OT and CRDT under the general transformation consistency framework. The shadowed blocks indicate common components shared by transformation consistency maintenance solutions for text editing.

| Work Flow | OT | CRDT |
|-----------|----|------|
| **Local User** | User A interacts with the local editor to generate a *position-based* $EO_{in}$ which takes effects on the $ES_{local}$ immediately and is given to the underlying LOH. | | |
| **LOH** | $LOH(EO_{in}) \to O_i$: 1. **Timestamp** a position-based $EO_{in}$ to make $O_i$. 2. **Save** $O_i$ in the operation buffer. 3. **Propagate** $O_i$ to remote sites. | $LOH(EO_{in}) \to O_{id}$: 1. **Convert** a position-based $EO_{in}$ into an identifier-based $O_{id}$. 2. **Apply** local $O_{id}$ in the object sequence. 3. **Propagate** $O_{id}$ to remote sites. |
| **CP** | $O_i$ is *position-based*. | $O_{id}$ is *identifier-based*. |
| **ROH** | $ROH(O_i) \to EO_{out}$: 1. **Accept** a remote $O_i$ under certain conditions, e.g. *causally-ready*. 2. **Transform** $O_i$ against concurrent operations in the buffer to produce $O_i'$ and $EO_{out} (= O_i'$ without timestamp). 3. **Save** $O_i$ (or $O_i'$) in the buffer. 4. **Apply** $EO_{out}$ on $ES_{remote}$. | $ROH(O_{id}) \to EO_{out}$: 1. **Accept** a remote $O_{id}$ under certain conditions, e.g. *causally-ready*. 2. **Apply** remote $O_{id}$ in the object sequence. 3. **Convert** identified-based $O_{id}$ to position-based $EO_{out}$. 4. **Apply** $EO_{out}$ on $ES_{remote}$. |
| **Remote User** | User B observes the effect of the remote $EO_{out}$ on $ES_{remote}$, which has the same effect of $EO_{in}$ on $ES_{local}$ observed by User A. |

Also shown in Table 1, OT and CRDT solutions share the same general requirement for the CP component: an external *causally-ordered* operation propagation and broadcasting service, which may or may not involve a central server (see detailed discussions on this point in [74]). One
characteristic difference in the CP component is: the propagated operations are position-based in OT solutions, but identifier-based in CRDT solutions. In the core components LOH and ROH, OT and CRDT differ significantly, which are elaborated in the next section.

4.2 Direct and Indirect Transformations
Fundamentally, every transformation-based consistency maintenance solution must have a way to record the concurrency-impact information arising from concurrent user actions. In OT solutions, such information is recorded as a buffer of concurrent operations; external position-based operations are timestamped but the positional reference nature is preserved in internal operations. In CRDT solutions, concurrency-impact information is recorded in an internal object sequence, which maintains the effects of all (sequential or concurrent) operations applied to the document so far (plus those objects representing the initial characters in the document), and external position-based operations are converted into identifier-based internal operations. These differences are captured in the LOH component in Table 1.

For the ROH component, OT and CRDT use radically different methods to derive the transformed operation at a remote site. In OT solutions, when a remote position-based operation arrives, control algorithms process it against selected concurrent operations in the buffer one-by-one, and invoke transformation functions to do the transformation in each step. The actual transformation is based on a compare-calculate method, which compares numerical positions (using relations <, =, or >) between the two input operations, and calculates their positional differences (using arithmetic primitives + or −) to derive the position of an output operation, as illustrated in Fig. 1-(b).

In CRDT solutions (e.g. WOOT), when an identifier-based operation arrives at a remote site, it is first applied in the internal object sequence, then the transformed (position-based) operation is derived by using a search-count method, which searches objects in the sequence and counts the number of visible objects along the way, as illustrated for WOOT in Fig. 1-(c). The search-count method is clearly less efficient than the arithmetic compare-calculate method. Logoot-like CRDT solutions have no tombstones, so all objects correspond to visible characters in the external state and the search-count method can be realized using binary-search which is not the same as in WOOT, but with its own special issues, as specified in [79,81] and also discussed in detail in [73].

In summary, OT records the concurrency-impact information in a buffer of concurrent operations, and transforms position-based operations directly by selecting concurrent operations from the buffer, comparing and calculating positional differences between concurrent operations. In contrast, CRDT solutions record the effects of all (sequential and concurrent) operations in an internal object sequence, and transforms operations indirectly by converting a position-based operation into an identifier-based operation (and applying identifier-based operations in the object sequence) at a local site, and converting an identifier-based operation back to a position-based operation at a remote site, by applying identifier-based operations in the remote object sequence and then searching and counting visible objects in the internal object sequence.

5 DISCUSSIONS
5.1 The Hidden Transformation Nature of CRDT
When we put OT and WOOT solutions to the same co-editing example side-by-side in Fig. 1, it is clear that both solutions produce identical position-based operations: O2 = I(2, "c") is transformed and become O2 = I(1, "c") at User A, while O1 = D(1) is unchanged at User B. The reader can verify this by comparing Fig. 1-(c) (for WOOT) and Fig. 1-(b) (for OT). This is an intuitive example that shows WOOT indeed is an alternative to realizing the general transformation. Moreover, when we describe the CRDT approach under the general transformation framework in Table 1, the transformation nature of CRDT becomes clear as well.

Why was the transformation nature of CRDT not evident previously? We draw attention to Steps 3 and 4 in handling a remote operation in Fig. 1-(c) (and the same steps in the ROH component for CRDT in Table 1). These two steps play the roles in converting an identifier-based
operation into a position-based operation, and applying a position-based operation on the external state to ensure consistency. However, both steps are omitted in the description of WOOT [41] and WOOT variations: the final step of handling a remote operation ends at Step 2, after integrating the identifier-based operation into the internal object sequence. For the scenario in Fig. 1-(c), if Steps 3 and 4 were omitted, User B would still see the document as "abc" even after the remote operation $D(idb)$ has been integrated into the internal sequence, while User A would continue to see the document as "ae" after $f(c, idc, idb, ide)$ has been internally processed. In each case, the external documents visible by Users A and B are neither convergent nor intention preserving. It is clear that these steps are not mere implementation details, but crucial steps to ensure the correctness of a consistency maintenance solution for co-editing.

In WOOT [41], a value($S$) function was briefly mentioned and supposed to map the internal object sequence $S$ to the external state visible to the user. However, there was no hint on when and how the value($S$) function might be invoked to map the internal object sequence $S$ to the external document state, to accomplish the final effect of replaying a remote operation. To achieve real-time update of the external document, value($S$) should be invoked whenever a remote identifier-based operation is integrated into the internal object sequence. In principle, the value($S$) function could be implemented in two alternative ways. One is to derive a position-based operation and apply this operation to the external document, as illustrated in Fig. 1-(c). The other is to: (1) scan the internal object sequence to extract visible characters and generate a new sequence by character-wise concatenation, and (2) reset the external document state with the generated sequence of characters, which will have included the effect of the newly integrated remote operation. The second alternative is generally more expensive than the first one. One way or another, handling a remote operation must include the steps that change the external document visible to the user. We found these steps indeed manifested themselves in the documentation and/or implementation of CRDT-based co-editors built by practitioners [74].

Unfortunately, theoretic CRDT work missed not only these key steps, but also (and more critically) lost sight of the big picture of a co-editing system, and failed to recognize real issues associated with CRDT for co-editors [73,74].

### 5.2 Demystifying the Commutativity Property of CRDT

Missing the big picture of a co-editing system is a root of common myths and misconceptions about CRDT object sequences and operations. We clear up those misconceptions and demystify the commutativity property of CRDT below.

One misconception is that CRDT object sequences and identifier-based operations are native (or internal) to the editor. This leads to the illusion (a common misconception) that there is no need for position-based operations, let alone the need to convert them to/from identifier-based operations. Evidences from existing CRDT solutions suggest otherwise: for tombstone-based WOOT variations [4,40,41,80] and RGA [45], the conversion of local position-based operations into identifier-based operations was explicitly described, although the conversion of remote identifier-based operations back to position-based operations was omitted; for non-tombstone-based CRDT solutions, such as Logoot variations [79,81], the conversion of remote identifier-based operations to position-based operations was explicitly described, though the conversion of local position-based operations into identifier-based operations was obscured. Clearly, designers of these CRDT solutions were cognizant of the fact that CRDT identifier-based operations and object sequences were invented for consistence maintenance purposes, but not native to text editors. Moreover, in all existing CRDT-based co-editors, CRDT internal object sequences and identifier-based operations are not native but external to real editors, as revealed in [74].

While it is unquestionable that there exists no CRDT solution that is native to any editors, some may still argue there might be a possibility for CRDT object sequences and identifier-based operations to be adopted in future editors and co-editors. Unfortunately, the following insights from past co-editing research and practice suggest that CRDT object sequences and identifier-based operations are poor candidates as the native data structures and operations for text (or other) editors, and by deduction co-editors.
• First, data structures and operation models of text editors ought to be designed for effective and efficient support for standard text editing operations and user interactions. There exists substantial well-established prior art on how to create and optimize text editors that are performant (e.g. initial loading time, memory paging speeds, etc.) [9,12,23,75] – desirable properties that should be preserved in co-editors as well. However, CRDT object sequences and identifier-based operations were created for supporting CRDT-based consistency maintenance, without any concern for efficient support of standard text editing functionalities.

• Second, existing research has found that published CRDT object sequences, operations and manipulation schemes have high time and space complexities and various correctness issues for serving the intended consistency maintenance purpose (see detailed discussions in [73]); it is inconceivable to use them as the basis for supporting unintended functions in standard text (or other) editors.

• Last but not least, past co-editing research and experiences in building real world co-editors suggested that co-editors ought to be built by separating, rather than mixing, concerns about consistency maintenance from concerns about conventional editing functions, to allow for simplicity, modularity, and efficiency of both conventional editing functions and consistency maintenance solutions (discussed in [74]). In OT-based co-editors [1,55,57,60,61,68,82], for example, the choice of strategies (e.g. what native data structures or operation models to use) for implementing efficient document editing is completely left to application designers, and the support for real-time collaboration is orthogonal to and interfaced with the editing application by exposed abstract-data-type, which is, in the case of text editing, a sequence of characters [9,23]. The idea to mix data structures and operations devised for consistency maintenance (e.g. CRDT) within future editors is not supported, but in fact contradicted, by experiences and insights from co-editing research and real world applications (see [74]).

Closely related to the above misconception is the notion that CRDT makes concurrent operations natively commutative by design, whereas OT makes concurrent operations commutative after the fact [47,48]. The fact is, as revealed above, CRDT identifier-based operations are not native to editors, but only used within the CRDT object sequence, and have to be converted from/to position-based operations in order to make them commutative in the document visible to users. In contrast, OT solutions directly transform concurrent position-based operations to make them commutative on the text sequence visible to users. In fact, CRDT has to achieve the same OT commutativity (i.e. to make concurrent position-based operations commutative to co-editors) after the fact as well, albeit indirectly. The commutativity of identifier-based operations on the internal CRDT object sequence should not be confused with the commutativity of position-based operations on the external text document visible to users.

Demystifying the commutativity property of CRDT provides much-needed clarity about what CRDT really is and is not to co-editors, which in turn helps to bring out the real differences between OT and CRDT in achieving the same commutativity of position-based operations on the external text document visible to users – the real objective of consistency maintenance for co-editors. Unfortunately, the CRDT way of achieving this objective turned out to be far more complex than OT and error-prone, which is discussed in [73].

5.3 General Differences between OT and CRDT in Time and Space Costs

While both OT and CRDT have followed the same general transformation approach to co-editing, they have taken radically different approaches to realizing this general transformation, particularly in recording the concurrency impact information – an internal operation buffer (for OT) versus an internal object sequence (for CRDT), which have had fundamental impacts on the design and complexity of OT and CRDT solutions. Without diving into technical details of specific solutions (which are covered in [73]), we highlight some general and characteristic differences between OT and CRDT in time and space costs:

• Variables in Determining Time and Space Complexities. As OT records the concurrency impact information in an internal operation buffer, the time and space complexity of an OT
solution depends on a variable \( c \) (for concurrency) – the number of operations saved in the operation buffer and involved in transforming an operation. The value of \( c \) is related to concurrency but unrelated to the document contents; and \( c \) is often bounded by a small value, e.g. \( 0 \leq c \leq 10 \), for real-time sessions with a few users. In contrast, CRDT uses an internal object sequence to record the concurrency impact information, the time and space complexity of a CRDT solution depends on a variable \( C \) (for Contents) or \( C_t \) (for Content with tombstones) – the number of objects in the internal object sequence. The value of \( C/C_t \) is determined by the document contents but unrelated to concurrency; and \( C \) is typically an order of magnitude larger than \( c \), e.g. \( 10^3 \leq C \leq 10^6 \), for common text document sizes ranging from 1K to 1M characters, while \( C_t \) is much larger than \( C \) with the inclusion of tombstones. In real-time text co-editing, the following inequality commonly holds: \( C_t \gg C \gg c \), which have major impacts on the theoretic complexity and practical efficiency of OT and CRDT solutions [73].

- **Costs in Handling Sequential and Concurrent Operations.** An OT solution has no time and space cost for transformation when there is no concurrent operation (with \( c = 0 \)) as the internal operation buffer can be emptied with garbage collection\(^7\), whereas a CRDT solution bears similar time and space costs regardless whether operations are sequential or concurrent since all operations must be applied and kept in the internal object sequence (with costs determined by \( C \) or \( C_t \)), which can never be emptied\(^8\) unless the document itself is empty.

- **Costs in Handling Local and Remote Operations.** An OT solution has no transformation cost in handling local operations since a local operation can never be concurrent with any operation in the buffer, whereas a CRDT solution bears almost the same processing costs regardless whether an operation is local or remote since every operation has to be applied in the internal object sequence. The longer time the local operation processing takes, the less responsive the co-editor is to the local user.

- **Costs in Session Initialization.** At the start of a co-editing session, the operation buffer for an OT solution is empty, bearing no space and time cost in initialization, whereas the internal object sequence for a CRDT solution must be created to represent initial characters in the document\(^9\), which incurs space and time overhead at the initialization time and bears the cost during a whole session. Session initialization complexity and cost can make differences in co-editing session management and handling later-comers during a co-editing session [6,60].

Detailed time and space complexity comparison of representative OT and CRDT solutions, as well as their correctness issues, are addressed in [73].

### 6 CONCLUSIONS

Based on a comprehensive review and comparison of OT and CRDT for consistency maintenance in real-time co-editing and in building real world co-editors, we have made a number of discoveries, which contribute to the advancement of the state-of-the-art knowledge on collaboration-enabling technology in general, and on OT and CRDT in particular.

In this article, we have presented one major outcome from this study – a general transformation framework, which not only provides a common ground for describing, examining and comparing a variety of consistency maintenance solutions in co-editing (e.g. OT and CRDT solutions, among others), and also may inspire invention of new consistency maintenance solutions in the future.

Another significant outcome, reported in this article, is revealing previously hidden but critical facts about CRDT: CRDT is like OT in following the same general transformation approach to consistency maintenance in real-time co-editors; CRDT is the same as OT in making user-generated operations commutative after the fact, albeit indirectly; and CRDT operations are not

\(^7\) Operation garbage collection is commonly used in OT solutions and OT-based co-editors [11,36,43,51,53,54,55,60,61,66,67,77,78,84].

\(^8\) Tombstones can be removed as garbage in some tombstone-based CRDT solutions (e.g. RGA [45]), but not in others (e.g. WOOT variations [4,40,41,80]). However, tombstone collection does not address the object sequence overhead issue, which exists in all CRDT solutions [73].

\(^9\) Nearly all CRDT articles ignored the existence and impact of initial document contents in calculating the size of the internal object sequence (see detailed analysis in [73]).
natively commutative to text editors, but require additional conversions between CRDT internal operations and external editing operations. Revealing these facts helps demystify what CRDT really is and is not to co-editing, which in turn helps bring out the real differences between OT and CRDT — their radically different ways of realizing the same general transformation approach.

Without diving into technical details of specific OT and CRDT solutions, in this article, we have outlined a number of general differences between OT and CRDT, including time and space costs in initializing a co-editing session, handling concurrent and sequential operations, and handling local and remote operations, as well as key variables that determine the time and space complexities of OT and CRDT solutions: complexities of OT depend on a variable c (for concurrency) — the number of concurrent operations involved in transforming an operation; complexities of CRDT depend on a variable C (for Contents) or Ci (for Content with tombstones) — the number of objects maintained in the internal object sequence. In-depth analysis of these variables and their impacts on complexities and correctness of representative OT and CRDT solutions are presented in [73].

We hope discoveries from this work will help clear up common myths and misconceptions surrounding OT and CRDT, inspire new and fruitful explorations of novel collaboration techniques, and accelerate progress in co-editing and collaboration-enabling technology innovation and real world applications.

7 ACKNOWLEDGMENTS

This research is partially supported by an Academic Research Fund Tier 2 Grant (MOE2015-T21-087) from Ministry of Education Singapore.

REFERENCES

1. Agustina, Liu, F., Xia, S., Shen, H.F. and Sun. C. CoMaya: Incorporating advanced collaboration capabilities into 3D digital media design tools. ACM CSCW (2008), 5-8.
2. Agustina and Sun. C. Dependency-conflict detection in real-time collaborative 3D design systems. ACM CSCW (2013), 715-728.
3. Agustina and Sun. C. Operational transformation for real-time synchronization of shared workspace in cloud storage. ACM GROUP (2016), 61-70.
4. Ahmed-Nacer, Ignat, M. C.-L, Oster, G., Roh, H.-G. and Urso, P. Evaluating CRDTs for real-time document editing, ACM DocEng (2011), 103–112.
5. Andrè, L., Martin, S., Oster, G. and Ignat, C.-L. Supporting adaptable granularity of changes for massive-scale collaborative editing. IEEE CollaborateCom (2013), 50–59.
6. Begole, J., Rosson, M.B. and Shaffer, C.A. Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems. ACM TOCHI 6, 2 (1999), 95 – 132.
7. Boucheneb, B. and Imine, A. On model-checking optimistic replication algorithms. FORTE on Formal Techniques for Distributed Systems (2009), 73 – 89.
8. Briot, L. Urso, P. and Shapiro, M. High responsiveness for group editing CRDTs. ACM GROUP (2016), 51–60.
9. Crowley, C. Data structures for text sequences. Computer Science Department, University of New Mexico, 1996. http://www.cs.unm.edu/~crowley/papers/sds/sds.html
10. Davis, A., Sun, C. and Lu, J. Generalizing operational transformation to the standard general markup language. ACM CSCW (2002), 58-67.
11. Day-Richter, J. What’s different about the new Google Docs: Making collaboration fast. https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html
12. Drucker, Peter F. A brief glance at how various text editors manage their textual data. https://ecc-comp.blogspot.com/2015/05/a-brief-glance-at-how-5-text-editors.html
13. Ellis, C. A. and Gibbs, S. J. Concurrency control in groupware systems. ACM SIGMOD (1989), 399–407.
14. Fraser, N. Differential Synchronization. ACM DocEng (2009), 13-20.
15. Gentle, J. ShareJS: Collaborative editing in any app. https://github.com/josephg/ShareJS
16. Greenberg, S. and Marwood, D. Real time groupware as distributed system: concurrency control and its effect on the interface. ACM CSCW (1994), 207 – 217.
17. Grudin, J. Why CSCW applications fail: problems in the design and evaluation of organizational interfaces. ACM CSCW (1988), 85-93.
18. Gu, N., Yang, J. and Zhang, Q. Consistency maintenance based on the mark & retrace technique in groupware systems. ACM GROUP (2005), 264 – 273.
19. Gutwin, C. and Greenberg, S. The effects of workspace awareness support on the usability of real-time distributed groupware. ACM TOCHI, 6(3), 1993, 243-281.
20. Ignat, C. and Norrie, M.C. Customizable collaborative editor relying on treeOPT algorithm. ECSCW (2003), pp. 315–334.

21. Imine, A., Molli, P., Oster, G. and Rusinowitch, M. Proving correctness of transformation functions in real-time groupware. ECSCW (2003), 277–293.

22. Imine, A., Rusinowitch, M., Oster, G. and Molli, P. Formal design and verification of operational transformation algorithms for copies convergence. Theoretical Computer Science (2006), 351(2):167–183.

23. Laird, Avery. Text Editor: Data Structures. wwwaverylaird.com/programming/the%20text%2 Eddie/2017/09/30/the-piece-table.

24. Lamport, L. Time, clocks, and the ordering of events in a distributed system. ACM 21, 7 (1978), 558-565.

25. Mihai Letia, M., Preguica, N., Shapiro, M. CRDTs: Consistency without concurrency control. RR-6596, INRIA. 2009.

26. Li, R., Li, D. and Sun, C. A time interval based consistency control algorithm for interactive groupware applications. IEEE ICPADS (2004), 429-436.

27. Li, D. and Li, R. Ensuring content and intention consistency in real-time group editors. IEEE ICDCS (2004), 748–755.

28. Li, R. and Li, D. A landmark-based transformation approach to concurrency control in group editors. ACM GROUP (2005), 284–293.

29. Li, D. and Li, R. An approach to ensuring consistency in Peer-to-Peer real-time group editors. JCSCW 17, 5-6 (2008), 553–611.

30. Li, D. and Li, R. An admissibility-based operational transformation framework for collaborative editing systems. JCSCW 19, 1 (2010): 1–43.

31. Liu, Y., Xu, Y., Zhang, S. and Sun, C. Formal verification of operational transformation. Proc. of 19th International Symposium on Formal Methods, 2014. LNCS Vol. 8442, 432-448.

32. Lv, X., He, F., Cai, W., and Cheng, Y. A string-wise CRDT algorithm for smart and large-scale collaborative editing systems. Advanced Engineering Informatics (2017), 33: 397 -409.

33. Koch, M. and Schwabe, G. Interview with Jonathan Grudin on Computer-Supported Cooperative Work and Social Computing. Bus Inf Syst Eng. DOI 10.1007/s12599-015-0377-1. Published online: 03 March 2015.

34. MacFadden, M. The client stop and wait operational transformation control algorithm. Solute Consulting, San Diego, CA, 2013.

35. MacFadden, M., Agustina, Ignat, C., Gu, N. and Sun, C. The fifteenth international workshop on collaborative editing systems. Companion of ACM CSCW (2017) workshop program, 351-354. http://cooffice.ntu.edu.sg/sigce/iwces15/.

36. Nichols, D., Curtis, P., Dixon, M. and Lamping, J. High-latency, low-bandwidth windowing in the Jupiter collaboration system. ACM UIST (1995), 111-120.

37. Macioka, P., Jahns, K., Derni, M., and Klamma, R. Near real-time peer-to-peer shared editing on extensible data types. ACM GROUP (2016), 39–49.

38. Prakash, A. and Knister, M. A framework for undoing actions in collaborative systems. ACM TOCHI 1, 4 (1994), 295 –330.

39. Preguica, N., Marquès, J. M., Shapiro, M. and Letia, M. A commutative replicated data type for cooperative editing. IEEE ICDCS (2009), 395–405.

40. Oster, G., Urso, P., Molli, P. and Imine, A. Real time group editors without operational transformation. Research Report RR-5580, INRIA, May 2005.

41. Oster, G., Urso, P., Molli, P. and Imine, A. Data consistency for p2p collaborative editing. ACM CSCW (2006), 259–268.

42. Oster, G., Molli, P., Urso, P. and Imine, A. Tombstone transformation functions for ensuring consistency in collaborative editing systems. IEEE CollaborateCom (2006), 1-10.

43. Ressel, M., Ruhland, N. and Günzenhauser, R. An integrating, transformation-oriented approach to concurrency control and undo in group editors. ACM GROUP (1996), 288–297.

44. Ressel, M. and Günzenhauser, R. Reducing the problems of group undo. ACM GROUP (1999), 131–139.

45. Roh, H.-G., Jeon, M., Kim, J.-S. and Lee, J. Replicated abstract data types: Building blocks for collaborative applications. JPDC, 71, 3. (2011), 354–368.

46. Shao, B., Li, D., Lu, T. and Gu, N. An operational transformation based synchronization protocol for web 2.0 applications. ACM CSCW (2011), 563 –572.

47. Shapiro, M. and Preguica, N. Designing a commutative replicated data type. arXiv:0710.1784v1 [cs.DC] 9 Oct 2007.

48. Shapiro, M., Preguica, N., Baquero, C. and Zawirski, M. Conflict-free replicated data types. SSSDS (2011), 386–400.

49. Shen, H.F. and Sun, C. Flexible notification for collaborative systems. ACM CSCW (2002), 77 – 86.

50. Spiewak, D. Understanding and applying operational transformation. www.codecommit.com/blog/java/java/understanding-and-applying-operational-transformation.

51. Suleiman, M., Cart, M. and Ferrié, J. Serialization of concurrent operations in a distributed collaborative environment. ACM GROUP (1997), 435 – 445.

52. Suleiman, M., Cart, M. and Ferrié, J. Concurrent operations in a distributed and mobile collaborative environment. IEEE ICDE (1998), 36–45.

53. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems. ACM GROUP (1997), 425 – 434.

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

55. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. ACM TOCHI 5, 1 (1998), 63 – 108.
56. Sun, C. Optional and responsive fine-grain locking in Internet-based collaborative systems,” *IEEE TPDS*, 13, 9 (2002). 994-1008.
57. Sun, C. Undo any operation at any time in group editors. *ACM CSCW* (2000), 191-200.
58. Sun, C. Undo as concurrent inverse in group editors. *ACM TOCHI* 9, 4 (2002), 309 – 361.
59. Sun, C. Consistency maintenance in real-time collaborative editing systems. Talk and demo at Microsoft Research (Redmond, USA) in Feb 2003. Video: http://cooffice.ntu.edu.sg/coword/vods/lecture.htm.
60. Sun, C., Xia, S., Sun, D., Chen, D., Shen, H. and Cai, W. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM TOCHI* 13, 4 (2006), 531 – 582.
61. Sun, C. OTFAQ: operational transformation frequently asked questions. http://cooffice.ntu.edu.sg/otfaq.
62. Sun, C. Issues and experiences in designing real-time collaborative editing systems. Tech talk and demo at Google (Mountain View, USA). 17 Nov. 2008. Video: https://www.youtube.com/watch?v=84zqbXUQHc.
63. Sun, C. Operational transformation theory and practice: empowering real world collaborative applications. *ACM CSCW* (2011) tutorial. http://cscw2011.org/program/t8.html
64. Sun, C., Agustina, and Xu, Y. Exploring operational transformation: from core algorithms to real-world applications. *ACM CSCW* (2011) demo. http://cscw2011.org/program/demos.html
65. Sun, D., Xia, S, Sun, C. and Chen, D. Operational transformation for collaborative word processing. *ACM CSCW* (2004), 437 – 446.
66. Sun, D. and Sun, C. Operation context and context-based operational transformation,” *ACM CSCW* (2006), 279 – 288.
67. Sun, D. and Sun, C. Context-based operational transformation in distributed collaborative editing systems. *IEEE TPDS* 20, 10 (2009), 1454 – 1470.
68. Sun, D., Sun, C., Xia, S. and Shen, HF. Creative conflict resolution in collaborative editing systems. *ACM CSCW* (2012), 1411-1420.
69. Sun, C., Wen, H. and Fan, H. Operational transformation for orthogonal conflict resolution in collaborative two-dimensional document editing systems. *ACM CSCW* (2012), 1391 – 1400.
70. Sun, C., Xu, Y. and Agustina. Exhaustive search of puzzles in operational transformation. *ACM CSCW* (2014), 519-529.
71. Sun, C., Xu, Y. and Agustina. Exhaustive search and resolution of puzzles in OT systems supporting string-wise operations. *ACM CSCW* (2017), 2504 – 2517.
72. Sun, C. Some reflections on collaborative editing research: from academic curiosity to real-world application. *IEEE CSCWD* (2017), New Zealand, 10-17.
73. Sun, D., Sun, C., Agustina, Cai, W. Real differences between OT and CRDT in correctness and complexity for co-editors. Submitted to arxiv.org on May 2, 2019.
74. Sun, D., Sun, C., Agustina, Cai, W. Real differences between OT and CRDT in building co-editing systems and real-world applications. Submitted to arxiv.org on May 2, 2019.
75. Valdes, R. Text editors: algorithms and architectures, not much theory but a lot of practice. *Dr.Dobb’s J.*(1993),38-43.
76. Valdes, R. The secret sauce behind Google Wave. May 31, 2009. https://blogs.gartner.com/ray_valdes/2009/05/31/the-secret-sauce-behind-google-wave/.
77. Vidot, N., Cart, M., Ferrie, J. and Suleiman, M. Copies convergence in a distributed real-time collaborative environment. *ACM CSCW* (2000), 171 – 180.
78. Wang, D., Mah, A. and Lassen, S. Google wave operational transformation. http://www.waveprotocol.org/whitpapers/operational-transform.
79. Weiss, S., Urso, P. and Molli, P. Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks. *IEEE ICDCS* (2009), 404–412.
80. Weiss, S., Urso, P. and Molli, P. Wooki: a p2p wiki-based collaborative writing tool. *WISE* (2007), 503–512.
81. Weiss, S., Urso, P. and Molli, P. Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE TPDC* 21, 8 (2010), 1162–1174.
82. Xia, S., Sun, D., Sun, C., Shen, H.F. and Chen, D.: Leveraging single-user applications for multi-user collaboration: the CoWord approach. *ACM TOCHI* (2004), 162–171.
83. Xu, Y., Sun, C. and Li, M. Achieving convergence in operational transformation: conditions, mechanisms, and systems. *ACM CSCW* (2014), 505-518.
84. Xu, Y. and Sun, C. Conditions and patterns for achieving convergence in OT-based co-editors. *IEEE TPDC* 27, 3 (2016), 695-709.