Real Differences between OT and CRDT in Building Co-Editing Systems and Real World Applications

David Sun, Codox Inc., United States
Chengzheng Sun, Nanyang Technological University, Singapore
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, simplicity, 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. Why? Based on comprehensive review and comparison study on representative OT and CRDT solutions and working co-editors based on them, we present our main research discoveries in relation to this question and beyond in a series of three related and complementary articles.

In prior work, we have revealed the similarities of OT and CRDT in following the same general transformation approach in co-editors, and their real differences in correctness and complexity. In this article, we examine the role of building working co-editors in OT and CRDT research, and the consequential differences in the adoption of and choice between OT and CRDT in real world co-editors and industry products. 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 some myths surrounding system implementation and “peer-to-peer” co-editing. We hope the discoveries from this work help clear up common myths 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].

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 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

©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
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 real-time collaboration in 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 Docs\textsuperscript{1} [11], to emerging start-up products, such as Codox Apps\textsuperscript{2}.

A variety of alternative techniques for consistency maintenance in co-editors had also been explored in the past decades [14,16,18,41,42,72]. One notable class of techniques is CRDT\textsuperscript{3} (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 [74]), 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 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? We believe that a thorough examination of these questions is relevant not only to researchers 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 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 actually the same as OT in following a general transformation approach to achieving consistency in co-editors. Moreover, we have 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 [73], we have organized the outcome materials into three parts and presented them in a series of three independent and complementary articles.

\textsuperscript{1}https://www.google.com/docs/about/

\textsuperscript{2}https://www.codox.io

\textsuperscript{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.
1. **Real differences between OT and CRDT under a general transformation framework for consistency maintenance in co-editors** [73]. In 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** [74]. In this article, 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** (this article). 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, the conclusion gives a summary of major results and contributions presented in the series of three articles.

### 2 BUILDING CO-EDITORS IN RESEARCH

Building and experimenting with working co-editors have played a crucial role in advancing co-editing research for over two decades [1,2,13,36,43,53,55,58,59,60,62,64,65,68,82]. Designing and implementing co-editors have enabled researchers to identify topics that are truly relevant and important to co-editing, to uncover intricate technical issues that would otherwise go unnoticed by pure theoretical study, to motivate innovative technical solutions, to experimentally validate solutions from system perspectives, and to gain critical insights for deriving general principles and theories from co-editing practices, which in turn inspire and guide experimental exploration. The success in adoption of co-editing research in industry products owes in no small part to the research community’s persistent efforts in connecting theory with practice and in innovating co-editing system design and implementation solutions.

In [73] and [74], we have revealed that the similarities of OT and CRDT in following a common general transformation approach in co-editors, and their real differences in correctness and complexity. In this article, we examine the role of co-editing system implementation played in OT and CRDT research, and the consequential differences in the adoption of and choice between OT and CRDT in real world co-editors and industry products.
3 BUILDING CO-EDITORS BASED ON OT

3.1 The Role of Building Co-Editors in OT Research

3.1.1 Use Co-Editors as Research Vehicles

One primary goal of OT research is to invent innovative solutions for building useful and useable co-editors. While theoretic work around OT algorithms has been fundamental to this endeavor, the practice of building OT-based working co-editors has historically played a crucial and integral role in driving and shaping OT solutions.

The very first OT research publication detailed the design and implementation of a plain-text co-editor GROVE [13]. A succession of working co-editors, including DistEdit [38], Jupiter [36], JOINT EMACS [43], and REDUCE [53,55,58], were built by researchers to investigate both system and theoretical issues in co-editing. These experimental efforts revealed critical insights into the dOPT puzzle (a flaw in the first OT control algorithm named dOPT [13,54]), and eventually led to its resolution and establishment of the theoretic foundation for OT – a comprehensive set of transformation conditions and properties, such as context-based conditions and transformation properties [38,43,54,55,61].

Early OT-based co-editors served as research vehicles to investigate novel consistency maintenance techniques for co-editing plain-text documents, but placed little emphasis on the relevance to supporting real world applications that users may use daily for content creation [17].

3.1.2 Apply OT to Real World Co-Editors and the TA Approach

It was around the year 2000 that researchers began to investigate the possibility of extending OT from supporting plain-text documents to off-the-shelf productivity suites with complex document formats and comprehensive functionalities. The Transparent Adaptation (TA) approach is one representative work along the line of explorations [60,65,82].

The goal of the TA approach was to extend the basic OT to support complex applications and to convert single-user editors into co-editors, without changing the source code of the original applications. The concurrency-centric nature of OT ensures the core control algorithms to be generic and allows transformation functions to be extensible to new application domains. Leveraged on this OT property, the TA approach is able to handle a myriad of user interactions and complex data objects found in modern productivity applications for real-time co-editing. A set of diverse productivity applications were examined and successfully converted into co-editors, including Microsoft Word (CoWord [60,65,68,82]), PowerPoint (CoPowerPoint [60,65]), and Autodesk Maya (CoMaya [1,2]).

![Fig. 1. (a) A TA-based architecture for co-editors. (b) CoWord architecture. (c) CoPowerPoint architecture.](image)

A TA-based co-editor consists of three architectural components [60], as shown in Fig. 1:

1. Generic Collaboration Engine (GCE), which provides generic transformation capabilities (independent of specific OT algorithms or transformation function).
2. Collaboration Adaptor (CA), which bridges the GCE with a single-user application, effectively extending basic transformation functionalities to the target single-user application.
3. A Single-user Application (SA), which provides conventional editing interface features and functionalities to users and suitable API (Application Programming Interface) to the CA.
The TA architecture provides a general framework for separating and integrating conventional editing functionalities (embedded in existing or newly designed single-user editors) and collaboration capabilities (powered by OT and/or other techniques) in co-editors. With the TA approach and a reusable GCE, the task of building a new co-editor is reduced to building a new CA for bridging a single-user editor with the GCE, without reinventing a new editor (if an existing editor is used) or re-implementing an OT-based collaboration engine.

3.1.3 Extend OT for Supporting Complex Editing Applications

Building TA-based CoWord, CoPowerPoint and CoMaya systems was the stimuli behind a number of key leaps in OT technology innovation. The first innovation was the extension of the basic OT data model from a single linear addressing space (for text editors) to a tree of multiple linear addressing domains — XOTDM (eXtended OT data model), which can be used to model complex documents in rich-text editors, word processors, and digital media design tools [1,2,60]. Complementing the OT data model extension was the extension of the OT operation model from supporting primitive insert and delete operations to arbitrary complex application operations.

For an editing system capable of supporting $N$ application operations, one school of thought is to design $N \times N$ transformation functions, one function for transforming each pair of application operations. We label this approach as AOT (Application Operation Transformation). This AOT approach can work well for editors supporting a relatively small number of operations, e.g. a plaintext editor with two operations insert and delete, four pair-wise transformation functions will be adequate [10,13,55,61,71]. However, designing direct application transformation functions is challenging for editors when the number of operations grow and when the semantics of those operations become complex (such as those in Word, PowerPoint and Maya). In addition, because application operations are by nature application-specific, their transformation functions are not reusable across different applications, which means significant redesign and validation efforts are needed for supporting a new co-editing application.

An alternative approach to supporting arbitrary application operations, pioneered in the TA-based CoWord, consists of two parts: (1) a collection of transformation functions for a small number of primitive operations, e.g. insert, delete, and update; and (2) a collection of adaptation schemes that translate arbitrary application operations to/from primitive operations. We label this transformation approach as POTCOA (Primitive Operation Transformation and Complex Operation Adaptation). The merit of this approach is that transformation functions for primitive operations are relatively easy to design and reusable across different applications; the challenge with this approach lies in designing the adaptation schemes between application operations and primitive operations. Experiences in applying this approach across a spectrum of applications have shown that devising adaptation schemes between complex application operations and primitive operations are relatively easier and more robust than designing direct AOT functions among complex application operations. For detailed discussion on issues related to adaptation schemes in TA-based co-editors, the reader is referred to [1,60,65,82].

3.1.4 Interactions between Academia and Industry in OT Research

OT originated from an industry research lab (Microelectronics and Computer Consortium (MCC)) in 1989 [13]. A few years later, several research groups independently discovered and resolved a fundamental OT algorithmic flaw (i.e. the well-known dOPT puzzle) [54], which revived interests on OT and helped establish OT and co-editing technical research as a niche system area in CSCW.

Follow-up OT research activities were mostly conducted in universities and research institutes. In the next over one decade, significant advancements were made along the lines of OT functionality extensions (e.g. from consistency maintenance to group undo, operation compression, etc.), algorithm design, puzzle detection and resolution, correctness verification, and complexity analysis [74]. These academic OT works nevertheless attracted little interest from industry. OT work was confined within the academic world until researchers started to extend OT to real world applications, and successfully built working systems, such as CoWord, CoPowerPoint, and CoMaya. These non-toy systems demonstrated the relevance of OT to the real world, which helped bridge and reconnect academia and industry [33].
Subsequently, academia researchers and industry practitioners have interacted in a number of forms, including technical talks at industry labs (e.g. [59,62]), demos of working prototypes (e.g. [59,62,64]), and tutorials on co-editing technologies (e.g. [63]), which were attended by people from both universities and industry, at ACM CSCW conferences. The co-editing research community and members of industry have also jointly organized a series of co-editing workshops (e.g. [35]), to share experiences in building and using co-editing systems.

3.2 OT-based Industrial Co-Editing Products and Techniques

3.2.1 OT Adoption in Major Industrial Products

In 2009, Google announced adoption of OT as the core technology for supporting its real-time collaboration features in Google Wave [76,78]. Since then, there has been an explosion of interests in OT and co-editing systems from computer industry and open source software communities.

In 2010, OT was adopted in Google Docs [11] — a Web-based real-time collaborative office suite. Google Wave OT algorithms and protocols were handled over to Apache Software Foundation and open sourced under the name Apache Wave⁴. It had strong influences on a number of Web-based open-source OT software projects. One representative project is ShareJS⁵. Other notable OT-based co-editors include SubEthaEdit⁶, CKEditor⁷, Etherpad⁸, to name a few.

In recent years, cloud storage companies started to extend their storage and file synchronization services to offering new Web-based co-editing services on the files in their storage. These companies built their own OT-based rich-text co-editors, such as Dropbox Paper⁹ (2017), and Box Notes¹⁰ (2017). More recently, Tencent also integrated the OT-based rich-text co-editing capability into its cloud-based TAPD (Tencent Agile Product Development) environment (2018)¹¹.

Most commercial collaborative editing products, such as Google Docs, Dropbox Paper, and Box Notes, have been designed and implemented for collaborative editing from scratch: from document markups and formats, editing operations, interface features, to application-specific transformation functions to support co-editing. However, building feature-rich and robust co-editors from scratch involves major investment of engineering resources, which is unaffordable to many. Furthermore, these co-editors are often vertically integrated with content storage solutions, which lock users in specific cloud services. For example, users need to move and store their contents to specific repositories, such Google Drive, Dropbox, and Box, in order to use these co-editors.

3.2.2 Alternative Approaches to Building OT-based Co-Editors on the Web

In contrast to Google Docs and other Web-based co-editors built from scratch, Codox Apps were built without reinventing new editors for conventional editing functionalities or re-implementing core OT solutions for each editor in supporting real-time collaboration capabilities.

Codox supports real-time co-editing directly in existing Web-based applications, such as Gmai¹², Evernote¹³, WordPress¹⁴, Zendesk¹⁵, Wikipedia¹⁶, TinyMCE¹⁷, Quill¹₈, and Slate¹⁹, and retain functionalities and the "look-and-feel" of original applications. In each of Codox Apps, an

---

⁴ https://en.wikipedia.org/wiki/Apache_Wave
⁵ https://github.com/share
⁶ https://www.codingmonkeys.de/subethaedit/
⁷ https://ckeditor.com/collaborative-editing/
⁸ http://etherpad.org/
⁹ https://paper.dropbox.com/
¹⁰ https://www.box.com/notes
¹¹ https://www.tapd.cn/
¹² https://chrome.google.com/webstore/detail/wave-for-gmail-and-evernote/dggkchdpkdalbalmmhlmgmiafjacofigfh?utm_source=inline-install-disabled
¹³ https://evernote.com/
¹⁴ https://wordpress.org/plugins/wave-for-wp/
¹⁵ https://www.zendesk.com/apps/support/wave/
¹⁶ https://www.wikipedia.org/
¹⁷ https://codepen.io/dnus/pen/ELRNMO
¹⁸ https://codepen.io/dnus/pen/OnjaeN
¹⁹ https://codepen.io/dnus/pen/yZdrwX
application-specific adaptor is injected into the single-user editing environment, which bridges an existing editor and an OT-powered reusable Codox engine, as shown in Fig. 2.

Most OT-based Web co-editors have adopted the AOT approach in designing transformation function, such as those transformation functions designed in Google Wave [78] and Docs [11], CKEditor\(^2\)\(^{20}\), etc. In contrast, the Codox Apps have taken the POTCOA approach to supporting complex application operations. See Section 3.1.3 for pros and cons of AOT and POTCOA.

![Codox Real-Time Cloud Service](image)

Fig. 2. Example Codox App architectures

Despite various technical differences among Web-based co-editors, we observed one common system design approach, i.e. to separate conventional editing functionalities and collaboration capabilities in different functional components, and provide ways to integrate these components together in a co-editing system, which is similar to the TA approach (see Section 3.1.2) \(^{60}\).

For example, the Codox approach is clearly a Web-based version of the TA approach. ShareJS supports co-editor design by providing a package of OT-based co-editing capabilities, which can be integrated with separately designed single-user editors. QuillJS\(^2\)\(^1\) and SlateJS\(^2\)\(^2\) provide frameworks for separately designing single-user rich text editors (based on different data and operation models) and multi-user collaboration capabilities (based on OT and/or other techniques), and offer APIs to bridge the two parts together to create co-editors. These alternative approaches are complementary and have shown to be promising to realize the dream of making real-time co-editors commonplace on the Web in the near future.

Last decade has witnessed significant efforts from industry and open source communities in applying OT to numerous real world applications, with large-scale deployment. The theory and practice of OT is now driven by a vibrant community of academic researchers and industrial practitioners. A wealth of valuable experiences have been accumulated from building and using co-editing applications, which calls for comprehensive technical reviews of real world system approaches, issues and experiences in the future.

### 3.3 Architectures and Communication Topologies of OT-based Co-Editors

OT-based co-editors have been built with a variety of system architectures and communication topologies, e.g. a star-like communication topology among co-editing clients, with a central server for broadcasting message and with optional transformation capabilities, or a fully connected communication topology without a central server for message broadcasting or transformation. The choice of a particular system architecture and communication topology depends on multiple technical factors in co-editing, such as where to store and access shared documents for co-editing, how to manage co-editing sessions (e.g. starting, joining and quitting sessions), how to propagate co-editing messages, and what kind of OT solution is used in the co-editor. In this context, whether the OT solution is a server-based or distributed solution is particularly relevant.

A server-based OT solution requires an OT-based server and runs different OT algorithms at the server and client sites in a co-editing session, as shown in Fig. 3. (a). Example server-based OT solutions include those in Jupiter \([36]\), Google Wave or Docs \([11,34,78]\), NICE \([49]\), CKEditor, ShareJS, QuillJS, and SlateJS. A distributed OT solution does not require an OT-based server and the same OT algorithm run at all co-editing sites, as shown in Fig. 3. (b) and (c). Representative

\(^{20}\) https://ckeditor.com/blog/Lessons-learned-from-creating-a-rich-text-editor-with-real-time-collaboration/

\(^{21}\) https://github.com/quilljs/delta

\(^{22}\) https://github.com/aha-app/collaborative-demo
distributed OT solutions include adOPTed [43], GOT [53,55], GOTO [54], TIBOT [26,84], COT [66,67], POT[84], SOCT2 [51] and SOCT3/4 [77] (though a server is used to issue continuous total ordering numbers but this server does not perform any transformation).

While server-based OT solutions require a server-based system architecture (for running the server part of OT solutions), distributed OT solutions do not dictate the choice of specific architectures or communication topologies for co-editors based on them. If a co-editor has adopted a distributed OT solution, it has various communication topology options, e.g. to use a message server with a star-like communication topology for broadcasting messages and other purposes (see Fig. 3. (b)), or a fully connected communication topology for message broadcasting without involving a server (see Fig. 3. (c)). The mere existence of a server in an OT-based co-editing system does not imply the server is necessarily required by the OT solution. Example co-editors based on distributed OT solutions include JOINT EMACS [43], REDUCE [53,55,58], CoWord [60], CoMaya [1,2], and Codox Apps.

![Fig. 3. Varieties of OT architectures and communication topologies. (a). Server-based OT (SOT) solution using a server for OT and communication (e.g. OT in Google Wave or Docs). (b). Distributed OT (DOT) solution using a server for communication (e.g. OT in CoWord); (c) Distributed OT (DOT) solution with full-connection for communication (e.g. OT in REDUCE).]

In reality, existing co-editors have commonly used servers to support shared document storage, session management, co-editing message broadcasting, and optionally for running server-side OT algorithms if a server-based OT solution is used. To our best knowledge, there has been no working co-editor based on OT (or other techniques) that does not use a central server, but the server is often used for supporting certain aspects of co-editing, not necessarily due to the need of OT.

## 4 BUILDING CO-EDITORS BASED ON CRDT

### 4.1 The Role of Building Co-Editors in CRDT Research

CRDT research for co-editing, started as an effort to solve the FT puzzle in OT without using OT algorithms (e.g. WOOT as the first CRDT), has adopted predominantly theoretic approaches to identifying co-editing issues, designing and verifying solutions (e.g. using theorem provers, model checkers, or mathematic proofs) [7,21,22,40,41,42], but rarely implemented proposed CRDT solutions in working co-editors for validation. These approaches have had profound impact in shaping CRDT research and on the correctness and complexity issues in CRDT solutions [74].

### 4.2 CRDT-based Co-Editors Built by Practitioners

We found no working CRDT-based co-editors built by CRDT researchers or academic literature documenting system experiences in using proposed CRDT solutions for building working co-editors. We did however find a dozen of CRDT-based co-editor projects hosted on GitHub, which were created by practitioners who were interested in learning whether and how CRDT solutions actually worked when applied to realistic editing environments.

#### 4.2.1 Observations on CRDT-based Plain Text Co-Editors

Most of those implementations are at rudimentary stages of development, but we found two relatively stable plain-text co-editing prototypes: Teletype\(^\text{23}\), which is based on WOOT (with

\(^{23}\)https://github.com/atom/teletype\)
tombstones) [8,40,41], and Alchemy Book24, which is based on Logoot (without tombstones) [79,81]. Based on our experimentation with live demos and review of available documentation and source code of those prototypes hosted at GitHub, we can make a number of observations.

First, CRDT-based co-editors were mostly developed by combining a CRDT solution with an existing text editor. For example, Teletype was developed by integrating WOOT with a desktop text editor named Atom25; Alchemy Book was built by integrating Logoot with a Web-based text editor named CodeMirror26. The implementations of Teletype and Alchemy Book show similarities with the TA approach [60]. These similarities are unsurprising since the TA approach is based on the general transformation approach but independent of specific transformation solutions, and CRDT has been shown to be an instance of the general transformation approach [73]. Those CRDT-based co-editors also show that CRDT internal object sequences and identifier-based operations are not native but external to real editors.

Second, CRDT-based co-editor implementations revealed key steps missed in CRDT publications. We examined how co-editing is supported end-to-end under CRDT-based co-editors, i.e. from the point when a user generates an operation from a local document, all the way to the point when this operation is applied to a remote document seen by another user. Both Teletype and Alchemy Book, as well as other CRDT-based co-editor implementations, unmistakably convert user-generated position-based operations into identifier-based operations at local sites (which is obscured in Logoot [79,81]), and convert remote identifier-based operations, after applying them in internal object sequences, to position-based operations at remote sites (which is ignored in WOOT [8,40,41]) and RGA [45]). This observation confirms our illustration in Fig. 1-(c) in [73], and description of CRDT under the general transformation framework in [73]. These missing steps are not mere implementation details, but crucial steps for CRDT solutions to achieve consistency maintenance in co-editors. 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, thus failing to recognize the nature of CRDT commutativity and real issues, which are discussed in detail in [73,74]. The moral of the story is that it is essential to build working co-editors to know what the real issues for co-editing are and experimentally validate theoretic solutions. Without the light shed from building real co-editors, theoretic exploration in the co-editing space could be lost in darkness and trapped in pitfalls or illusions, leading to nowhere.

Third, CRDT-based co-editors have all used a central server to support some aspects of co-editing. For example, Alchemy Book uses a central server for session management and broadcasting messages among co-editing clients, which is similar to OT-based CoWord [60]; Teletype uses a central server for client-discovery (session management) but allows co-editing clients to be fully connected for broadcasting messages without involving the server, which is similar to OT-based REDUCE [55,58]. To our best knowledge, there has been no single example of CRDT-based co-editors that was built without using a client-server architecture. The often-suggested idea that CRDT solutions are specifically designed for peer-to-peer collaborative editing [5,41,45,79,80,81] is tenuous at best, and confounds what CRDT solutions like WOOT and Logoot actually do. We further elaborate this point in Section 5.2.

By experimenting with Teletype and Alchemy Book prototypes, we can confirm the analytical results about tombstone-based (WOOT) and non-tombstone-based (Logoot) CRDT variations, discussed in [74]. In Teletype, we experienced tombstone overhead effects, where the co-editor suffered significant increase of memory consumption and degradation of performance, in both local response and remote replay, as the number of deletions increases during a session. In Alchemy Book, we were able to produce concurrent-insert-random-interleaving results when performing concurrent insertions at the same location, and experienced document inconsistencies under numerous scenarios (see concrete examples in [74]). The concurrent-insert-interleaving issue was also independently reported by a developer who tried to implement Logoot for text co-editing27.

24 https://github.com/rudi-c/alchemy-book.
25 https://atom.io/
26 https://codemirror.net/
27 https://stackoverflow.com/questions/45722742/logoot-crdt-interleaving-of-data-on-concurrent-edits-to-the-same-spot.
4.2.2 Extending CRDT for Rich Text Co-Editing

While most existing CRDT solutions for co-editors are confined to object sequences for supporting plain-text documents, there are some efforts towards extending CRDT to rich-text co-editing. Among those GitHub-hosted projects, the rich-text extension of Yjs\(^28\) — a WOOT-like tombstone-based CRDT solution [37] — is a representative example.

In fact, the Yjs extension did not change the core CRDT object sequence and identified-based operations, or made the CRDT objects and operations native inside a rich-text editor. Instead, the Yjs extension introduced an extra layer of rich-text-aware object sequence and operations (of various kinds), which have the knowledge about rich-text features (e.g. bold, italic attributes), but do not contain tombstones. This extra layer is placed between the Yjs core CRDT solution and the API of a separate rich-text editor (e.g. QuilJS). Consequently, there exist two object sequences outside a rich-text editor: one is the rich-text object sequence (without tombstones); and the other is the Yjs core CRDT object sequence (with tombstones but without the knowledge of rich-text).

Converting schemes are devised to resolve data and operation differences among multiple layers, e.g. data objects and operations from the target rich-text editor API are converted to/from objects and operations defined on the intermediate rich-text-aware layer (without tombstones), which in turn are converted to/from the CRDT objects (with tombstones) and identifier-based operations in the Yjs core layer. Clearly, main challenges to the Yjs extension lie in the correctness and efficiency of those multiple layers and conversion schemes.

Again, we observed conceptual similarities between the Yjs extension method and the TA approach (note: we made similar observations for CRDT plain-text co-editors like Teletype and Alchemy Book in Section 4.2.1), as well as their technical differences in complexity and efficiency. The success or failure of a TA-based co-editor is critically dependent on the capability, correctness and complexity of the core solution (e.g. an OT solution or the Yjs CRDT core solution) and the bridging techniques (e.g. an OT-based collaboration adaptor or the Yjs intermediate data and operation models and conversion schemes) between the core solution and the target editor. The Yjs extension is in an early stage of development and its viability remains to be seen.

4.3 CRDT-based Co-Editing Products

Among existing CRDT-based co-editors, Teletype for Atom is sometimes cited in the CRDT community as an example of industrial adoption of CRDT in co-editors. Apart from Teletype, we are not aware of any other industry co-editing product that is based on a CRDT solution.

Why CRDT solutions were rarely adopted in industrial co-editing products? Apart from the CRDT correctness and complexity issues discussed in [74], another obstacle to CRDT adoption in the real world, in our view, is that most CRDT solutions for co-editors are confined to inserting and deleting characters or objects in a linear sequence, which are clearly inadequate for supporting real world applications, e.g. rich text formatting, itemization, tables, images, etc. These real world application features have become de facto requirements for collaborative content creation and been commonly supported by OT-based co-editing products, e.g. Google Docs, Dropbox Paper and Codox Apps. Works on extending CRDT for rich text co-editing, e.g. the Yjs extension, were intended to address this limitation, but still in early stages of development.

5 DISCUSSION ON IMPLEMENTATION AND PEER-TO-PEER CO-EDITING

5.1 Myths and Facts about OT and CRDT Implementation

Despite the facts that OT solutions have been commonly implemented and adopted whereas CRDT solutions are rarely seen in working co-editors, there have been some often-heard claims: OT is complex to implement, and CRDT is simple to implement. We share some of our experiences in implementing OT and OT-based co-editors, and refute these claims below.

\(^28\) https://github.com/y-js/ The core Yjs solution is based on WOOT but made extensions and changes to WOOT, including a garbage collection scheme for removing tombstones “after a fixed time period” [37]. Those changes intended to reduce the time and space complexity of WOOT, but unfortunately incurred correctness issues (e.g. the garbage collection scheme) within the core Yjs. Detailed elaboration of those issues is beyond the scope of this article.
Building co-editors based on OT is no doubt challenging, and could be time-consuming for people without suitable background knowledge and system experiences. In over two decades, we have designed and implemented a range of OT solutions, from generic control algorithms GOT [53,55], GOTO [54,57,58], NICE [49], COT [66,67], and POT [83,84], to transformation functions for a variety of applications [2,3,10,55,61,65,69,70,71]. Moreover, we have built over a dozen of OT-based co-editing systems and products, including the Web-based pain-text co-editor REDUCE [55,57], desktop productivity tools CoWord and CoPowerPoint [60,65,82], digital design tools like CoMaya [1,2], and a range of Web-based Codox Apps (see Fig. 2 and Section 3.2).

From those experiences, we have learnt that the bulk of co-editor implementation challenges lie mostly in applying OT solutions to build a complete co-editing system, rather than in implementing OT algorithms themselves, which are at the core but only part of a co-editing system. Implementing OT algorithms inside working co-editors is non-trivial to people without prior knowledge of OT and co-editing, but should become significantly easier if one has done the work properly at least once and understood what had been done.

OT was reported to be hard and time consuming to implement in a well-known quote by a former Google Wave engineer29:

"Unfortunately, implementing OT sucks. [...] The algorithms are really hard and time consuming to implement correctly. [...] Wave took 2 years to write and if we rewrote it today, it would take almost as long to write a second time."

The above quote was often used by some people to prove OT implementation is complex and indirectly argue for CRDT simplicity, by following the logic – what is hard for OT must be easy for CRDT as CRDT works without OT. However, what is less known and ignored is that the same engineer later amended the previous comments with the following30:

"For what its worth, I no longer believe that wave would take 2 years to implement now - mostly because of advances in web frameworks and web browsers. When wave was written we didn't have websockets, IE9 was quite a new browser. We've come a really long way in the last few years."

The above amended statements revealed that major challenges of Google Wave were due to the Web frameworks, browsers, and communication utilities, etc. which were chosen to build the OT-based Google Wave, rather than merely implementing the core Google Wave OT algorithms [36,78]. This reflection is consistent with our experiences and with the fact that OT solutions have been implemented and adopted in numerous real world co-editors.

On the other hand, the notion that CRDT is simple to implement was never substantiated by evidence, but only contradicted by the facts that CRDT implementations in working co-editors are rarely seen and robust CRDT implementations are virtually nonexistent.

5.2 Myths and Facts about Peer-to-Peer Co-editing

As pointed out before, all known co-editors have used the client-server model for supporting certain aspects of co-editing, e.g. discovering and tracking users in a session, and/or editing operation broadcasting. In spite of this fact, CRDT authors have often used "peer-to-peer collaborative editing", or "p2p co-editing" in short, as one primary differentiator between CRDT and OT [5,37, 41,45,79,80,81]. We have found that the term of "p2p co-editing" in existing literature is rather ambiguous and often conflates multiple factors. In the following subsections, we will tease apart these factors and discuss their relationships to OT and CRDT solutions.

5.2.1 What Constraints Imposed on Operation Propagation and Communication?

Generally, both OT and CRDT solutions require editing operations to be executed in their causal-effect orders (based on the happen-before relation [24]) at all co-editing sites, to meet the causality-preservation requirement for co-editors [13,54,55]. This causal ordering can be achieved by adopting any suitable distributed computing techniques (based on either client-server or peer-to-peer models), which are orthogonal to OT or CRDT.

---

29 https://www.championtutor.com/7004/ (Nov 6, 2011)
30 https://news.ycombinator.com/item?id=12311984 (Aug 18, 2016)
However, some CRDT solutions, like WOOT, replaced the general causal ordering condition with WOOT-specific execution conditions: an operation can be accepted for execution only if the two neighboring objects (for an insert) or the target object (for a delete) already exist in the internal object sequence. WOOT-specific execution conditions were quoted as a merit for supporting p2p co-editing [41], but such conditions cannot ensure causality-preservation generally required for co-editors [13,54,55], and they are costly to check with the time complexity \(O(C_i)\). Other CRDT solutions (e.g. RGA and Logoot variations) and OT solutions are all adopting the general causal ordering condition. Therefore, with the exception of WOOT variations, causally-ordered operation propagation and execution are common requirements for OT and CRDT solutions, and hence, in general, not differentiating factors between OT and CRDT.

### 5.2.2 Whether Any Server is Required?

CRDT solutions do not explicitly require a communication server, but all, except WOOT, assume the existence of an external causal-order-preserving communication service.

On the other hand, some OT solutions, notably Jupiter [36], NICE [49], and Google Wave and Docs [11,78], explicitly require a transformation-based server to do part of the transformation and to broadcast operations (they are server-based OT solutions discussed in Section 3.3); SOCT3 and SOCT4 [77] require a server to issue continuous total ordering numbers for labelling operations.

However, most other OT solutions, such as adOPTed [43], GOT[53,55], GOTO[54], SOCT2 [51], TIBOT [26,84], COT [66,67], and POT[84], do not require a central server to do (any part of) the transformation work (they are distributed OT solutions discussed in Section 3.3), but only require the use of an external causal-order-preserving communication service, which is essentially the same as most CRDT solutions. Therefore, whether or not requiring a server is merely a feature of individual solutions, but not a general differentiating factor between OT and CRDT either.

### 5.2.3 Vector-based or Scalar Timestamps?

Yet another factor often cited in connection to p2p co-editing is whether vectors (with one element for each of co-editing sites in a session) or scalars (with a fixed number of variables) are used for timestamping or control in OT or CRDT solutions. Again, both vector-based and scalar-based timestamps and control have been used in OT and CRDT solutions.

Table 1. Summary of p2p co-editing factors and their relationships with OT and CRDT solutions

| P2P Co-Editing Factors | OT | CRDT |
|------------------------|----|------|
| Requirement for a Central Server | Yes | Some OT solutions (e.g. Jupiter [36], NICE [49], Google Wave & Docs [11,78]) require a transformation-based server; some (e.g. SOCT3 & SOCT4 [77]) require a server to issue continuous total ordering numbers. |
| | No | Many OT solutions (e.g. adOPTed [43], GOT[53,55], GOTO[54], SOCT2 [51], TIBOT [26,84], COT [66,67], POT[84]) only require the existence of an external causal-order-preserving communication service. |
| Requirement for Operation Propagation or Reception | Causal Ordering | All OT solutions |
| | Non-causal Ordering | Non |
| Timestamping Schemes | Vector | Some OT solutions (e.g. adOPTed [43], GOT[53,55], GOTO[54], SOCT2 [51], COT [66,67]) use vector-based timestamping. |
| | Scalar | Many OT solutions (e.g. Jupiter [36], NICE [49], Google Wave & Docs [11,78], SOCT3 & SOCT4 [77], TIBOT [26,84], POT[84]) use scalar-based timestamping. |
| | | CRDT solutions (except WOOT) do not explicitly require a communication server, but assume the existence of a causal-order-preserving communication service. |
| | | OT solutions (including WOOT) do not require a server to issue continuous total ordering numbers for labelling operations. |
| | | All CRDT solutions except WOOT. |
| | | WOOT [41] and variations [4,80] |
| | | OT solutions (except WOOT) require the existence of an external causally-ordered broadcasting service, without specifying what meta control information (or timestamping) needs to be provided to such external service to achieve causally-ordered broadcasting. |
For example, the RGA solution uses vector-based timestamps to reduce the time complexity from $O(C^2)$ (in WOOTH [4] and WOOTO [80]) to $O(C)$ and for garbage collection of tombstones (further reduce $O(C)$ to $O(C)$); Logoot [79] and variations [5, 81] use variable length (bounded by the object sequence length $C$) object identifiers, and require an external causally-ordered broadcasting service. On the other hand, some OT solutions, including adOPTed [43], GOT [53, 55], GOTO [54], and COT [66,67], used vector-based timestamps, but other OT solutions, including Jupiter [36], NICE [49], TIBOT [26,84], and SOCT4 [77], Google Wave and Docs [11,78], and POT [84], used scalar-based timestamps. In fact, scalar-based timestamping had been introduced to OT solutions long before the first CRDT (WOOT) came into being. Therefore, neither CRDT nor OT is unique in using vector-based and scalar-based timestamps or control.

In summary (see Table 1), all relevant p2p co-editing factors are orthogonal to OT and CRDT, namely whether or not requiring a central server, causally ordered communication, or vector/scalar timestamping, is merely a feature of individual solutions, rather than a general differentiator between OT and CRDT. The notion that CRDT is more suitable than OT in supporting p2p co-editing is a fallacy. Up until now, there has been no single p2p co-editor built yet; it remains open whether and how to support p2p co-editing in the future.

6 CONCLUSIONS
In a series of three independent and complementary papers, including this one and [73,74], we have reported our discoveries from a comprehensive review and comparison of OT and CRDT for consistency maintenance in real-time co-editing and in building real world co-editors. These research outcomes contribute to the advancement of the state-of-the-art knowledge on collaboration-enabling technology in general, and on OT and CRDT in particular. We summarize our main discoveries and contributions reported in three articles below.

6.1 A General Transformation Approach and What CRDT Really Is and Is Not [73]
One significant outcome of this study is the discovery of a general transformation approach, which not only provides a common ground for describing, examining and comparing a variety of concurrency control solutions in co-editing (e.g. OT and CRDT solutions, among others), and also may inspire invention of new consistency maintenance solutions in co-editing in the future.

Another significant outcome is revealing previously hidden but critical facts about CRDT: (1) CRDT is like OT in following the general transformation approach to consistency maintenance in real-time co-editors; (2) CRDT is the same as OT in making user-generated operations commutative after the fact, albeit indirectly; and (3) CRDT operations are not 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, and in turn brings out the real differences between OT and CRDT for co-editors – their radically different ways of realizing the same general transformation approach.

6.2 Algorithmic Differences in Correctness and Complexity [74]
Key insights from probing what really differentiates OT and CRDT include: (1) OT is concurrency-centric in the sense it treats generic concurrency issues among operations as its first priority at the core control algorithms, and isolates the handling of application-specific data and operation modelling issues in transformation functions; (2) CRDT is content-centric in the sense that it takes the first priority to manipulate internal application-related contents, including object sequences and schemes for searching and applying identifier-based operations in the object sequence, but mixes the handling of concurrency issues within object search and manipulation schemes. This concurrency-centric vs content-centric difference is fundamental and has profound implications to OT and CRDT solutions.

The first significant implication is in the different design and correctness issues in OT and CRDT solutions. Key OT design issues include designing control algorithms to deal with generic concurrency issues, and designing separate transformation functions to handle application-specific issues. OT-special challenges and puzzles, such as ensuring context-based conditions (e.g. the
The $dOPT$ puzzle was a case of violating the context-equivalence condition, and avoiding or preserving CP2 (e.g. the $FT$ puzzle was a case of violating of the CP2 property), were derived from and all solved under the concurrency-centric approach. The correctness of key OT components, including generic control algorithms and transformation functions for a range of commonly used operation and data models (e.g. string-wise plain-text editing and beyond), has been established under well-defined conditions and properties.

In contrast, key CRDT design issues include designing CRDT-special data structures and schemes for representing and manipulating object sequences, searching and executing identifier-based operations in the object sequence, and conversions between internal identifier-based operations and external position-based operations, which collectively deal with both application-specific and concurrency issues in co-editing. This approach has induced CRDT-specific challenges and puzzles, such as tombstone overhead, variable and lengthy identifiers, and the correctness of CRDT key data structures and functional components. In this work, we have detected multiple correctness problems with Logoot: inconsistent-position-integer-ordering and infinite loop flaws, position-order-violation puzzles, and concurrent-insert-interleaving puzzles. It remains an open challenge to resolve these issues under the CRDT approach to co-editing. Contradicting the common claims that CRDT solutions were simple and obvious in correctness, the correctness of key components in various CRDT solutions, e.g. object identifiers and sequences, and object sequence searching and manipulation schemes, remains to be verified and validated, using well-defined criteria, which are yet to be established as well.

The second significant implication is found in the different time and space complexities among OT and CRDT solutions. OT complexity is determined by a variable $c$ (for concurrency) – the number of concurrent operations involved in transforming an operation; CRDT complexity is dominated by a variable $C$ (for Contents) or $C_t$ (for Content with tombstones) – the number of objects maintained in the internal object sequence. The different natures of variables $c$ and $C/C_t$ have had profound impact on the time and space complexity of OT and CRDT solutions.

Theoretically, representative OT solutions have achieved the time complexity $O(1)$ for processing local operations, and $O(c)$ or $O(c^2)$ for processing remote operations; and the space complexity $O(c)$, $O(c-m)$, or $O(c-m^2)$, where $m$ is the number of real-time co-editing users in a session (usually $m < 5$), under various OT system architectures and protocols. In contrast, representative CRDT solutions have the time complexity ranging from $O(C^4)$, $O(C^2)$, to $O(C)$ (for tombstone-based solutions), or $O(C \cdot \log(C))$ (for non-tombstone-based solutions); and the space complexity ranging from $O(C)$ (for tombstone-based solutions, without tombstone garbage collection) or $O(C)$ (for tombstone-based solutions, with tombstone garbage collection), to between $O(C)$ and $O(C^2)$ (for non-tombstone-based solutions). In terms of time and space complexity, $C/C_t$-based CRDT solutions are evidently more complex than $c$-based OT solutions (see details in Table 3 in [74]).

In addition to the theoretic complexity differences, we highlight the practical differences of the input variables in those complexity expressions: $c$ is often bounded by a small value, e.g. $0 \leq c \leq 10$, for a real-time session with a few (e.g. less than 5) users; $C$ is orders of magnitude larger than $c$, e.g. $10^3 \leq C \leq 10^6$, for common plain text document sizes ranging from 1K to 1M characters, while $C_t$ could be much larger than $C$ due to the inclusion of tombstones. In real-time text co-editing, the following inequality commonly holds: $C_t \gg C \gg c$. It remains an open challenge to devise $C/C_t$-based CRDT solutions that can match $c$-based OT solutions in time and space complexity and in practical performance.

The third implication is in the generality and extendibility of OT and CRDT solutions for co-editors, which are covered in both [74] and this article of this series. OT solutions separate generic concurrency issues from application-specific data and operation issues, with the core control algorithms being generally applicable to different application domains beyond text editing. Extensions of existing OT solutions can be and have been achieved by designing new transformation functions and adaptation schemes for new applications (e.g. word processing and digital media design), without reinventing its core control algorithms. In contrast, CRDT solutions mix concurrency issues with application-specific data and operation issues, with key CRDT
components being intricately related to each other and coupled with application-specific object sequences and operations. So far, most CRDT work for co-editing has been confined to plain-text editing; extensions for supporting rich-text editing are at early stages of exploration.

6.3 Differences in System Implementation, Validation and Real World Application

In the third article of the series, we have examined the role of building working co-editors in OT and CRDT research, and found the major differences between OT and CRDT in implementation and validation: OT solutions have been commonly implemented and validated in working co-editors, whereas CRDT solutions were rarely implemented and validated in working co-editors. In particular, we have reviewed the evolution of co-editors from research vehicles to real world applications, discussed representative OT-based co-editors and made observations on alternative approaches in industry products and open source projects, and described server-based versus distributed OT solutions and their relationships with system architectures and communication topologies of co-editors based on them. Moreover, we have discussed our observations on CRDT-based co-editors in relation to published CRDT solutions, clarified some myths surrounding system implementation complexity, and pointed out the fallacies in the notion that CRDT is simple to implement and more suitable than OT for peer-to-peer co-editing.

6.4 Summary and Future Work

In summary, we have learned from this work that CRDT is to address the same consistency maintenance problems in co-editors and to meet the same consistency requirements as OT; CRDT is like OT in following the same general transformation approach; and CRDT is not natively commutative to co-editors. Furthermore, we have revealed the real differences between OT and CRDT in correctness, time and space complexity, extendibility, implementation and validation, which we believe are the key factors that have affected the adoption of and choice between OT and CRDT for co-editors in the real world. This work has presented ample facts and evidences that refute CRDT superiority claims over OT on all accounts.

Numerous alternative solutions for consistency maintenance in co-editors have been explored in the past research, and a wealth of experiences and lessons have been accumulated from those explorations. It is high time to critically review those alternatives and to question: what an alternative approach really is to co-editing, what it has really achieved so far, whither it is heading, and what could be expected following its past trajectories. For any alternative to become a viable solution in co-editing, in our view, it should be able to offer capabilities that are truly superior to existing state-of-the-art solutions for the same purposes, and demonstrate the relevance and applicability in supporting real world co-editors.

Past decades have witnessed the evolution of co-editing research from a niche area in CSCW and distributed computing to a host of core collaboration-enabling techniques widely used in real world co-editors and major industrial co-editing products. Moving forward, we foresee new challenges and opportunities in bridging core co-editing techniques with emerging and co-editing applications, and in extending core techniques to application domains beyond co-editing.

We hope discoveries from this work will help clear up common myths and misconceptions surrounding OT, CRDT and other alternative co-editing approaches and techniques, 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. R., Roh, H.-G. and Urso, P. Evaluating CRDTs for real-time document editing, *ACM DocEng* (2011), 103–112.
5. Andre e, 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/~crawley/papers/dds/dds.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 copresence convergence. *Theoretical Computer Science* (2006), 351(2):167–183.
23. Laird, Avery. Text Editor: Data Structures. www.averylaird.com/programming/the%20text%20editor/2017/09/30/the-piece-table.
24. Lamport, L. Time, clocks, and the ordering of events in a distributed system. *CACM* 21, 7 (1978), 558-565.
25. Mihai Letia, M., Preguica, N., Shapiro, M. CRDTs: Consistency without concurrency control. *RR-6956, 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/wwes15/.
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. Nicolaescu, P., Jahns, K., Derntl, 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. Preguic, N., Marqués, J. M., Shapiro, M, and Letia, M. A commutative replicated data type for cooperative editing. IEEE ICDCS (2009), 395–403.
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., Ruhlman, N. and Gunzenhauser, R. An integrating, transformation-oriented approach to concurrency control and undo in group editors. ACM CSCW (1996), 288 – 297.
44. Ressel, M. and Gunzenhauser, 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 Zawirska, 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., Carl, M. and Ferrié, J. Serialization of concurrent operations in a distributed collaborative environment. ACM GROUP (1997), 435 – 445.
52. Suleiman, M., Carl, 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, C., Sun, D., Agustina, Cai, W. Real differences between OT and CRDT under a general transformation framework for consistency maintenance in 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 correctness and complexity for consistency maintenance in co-editors. 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/whitewpapers/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 CSCW* (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.