

# Achieving CCI Efficiently by Combining OT and Dynamic Locking with Lazy Consistency in a Peer-to-Peer CES

Jon A Preston and Sushil K Prasad

Department of Computer Science

Georgia State University

Atlanta, GA USA

jon.preston@acm.org and sprasad@gsu.edu

## ABSTRACT

This paper discusses our work in combining dynamic locking with operational transformation (OT) to achieve low computational and communication costs in maintaining a real-time, synchronous collaborative editing system. We discuss our approach in storing the document in a distributed, hierarchical tree and maintaining consistency, causality preservation, and intention preservation via a peer-to-peer caching system; our approach employs dynamic, hierarchical locking at various levels within the document's structure and also supports OT at atomic levels within the document when locking is not desirable. Because changes are sent to other users within the system only as needed (and cached when possible), our approach minimize communication costs among multiple readers and writers. We offer an overview of our techniques and their correctness, showing how users always have a most current/correct copy of the section(s) of the document which they are viewing.

## Keywords

Collaborative Editing System, CES, Operational Transformation, OT, Dynamic Locking, Distributed Systems, Cache Coherency

## INTRODUCTION

Collaborative editing systems (CES) and consistency maintenance within distributed, synchronous CES are active research areas within the field of computer supported collaborative work (CSCW) [15]. Ensuring consistency, causality preservation, and intention preservation – the CCI model [14][15] – is central to CES research.

Locking is one technique used to ensure consistency and data integrity in distributed systems, but locking has the disadvantage of reducing concurrent access. Consequently, the CES research community has adopted operational transformation (OT) or similar approaches to maintain real-time, high responsiveness while striving to maintain the CCI model. Motivated by the idea that locking and OT

serve complimentary roles [16], we incorporate both techniques such that users are able to make changes locally within locked (or partially-locked) sections of a document and adopt OT as needed (when many users make changes to a single section of the document).

We propose adopting a more relaxed consistency model in which not all users within the CES have the most current copy of the document – rather, all users have the most current copy of the section of the document they are viewing (i.e., the visible/focused portion of the document is always current on a user-by-user basis). By relaxing the consistency constraint within the CES, we are able to reduce communication and computation costs while at the same time improve the intention preservation of users.

The remainder of this paper is structured as follows: we begin by discussing how the document may be viewed as a hierarchical tree of content and how such a tree may be distributed in a peer-to-peer system, maintaining optimistic locks and tracking readers and writers; next, we present how consistency, causality, and intention constraints are enforced within the context of existing, well-formed OT approaches; finally, we provide conclusions and discuss our future work.

## HIERARCHICAL DOCUMENT TREE

Traditionally, research within CES has viewed documents to be a linear sequence of data; consequently, OT and other techniques to ensure the CCI model [15] are designed to work on linear content. More recently, others have proposed leveraging the semantic structure of the document and viewing it as a hierarchy [5][6][10]. Operations to ensure CCI are more efficient when applied to sections of a hierarchical document as opposed to the entire document, and the system is better able to handle context-specific consistency/intention preservation [5][16].

We extend this view of the document as a hierarchical structure; in addition to better achieving context-specific consistency preservation, we can reduce communication and computational costs. Based upon the semantic structure of the document, the document may be broken up into sections, subsections, paragraphs, sentences, words, etc. If the document being shared is a CAD drawing, it may be broken into layers, objects, etc. If the document is programming source code, it may be broken into classes,

components, methods, blocks, etc. Thus we do not have any preconceived notion of what the sections of the document contain, nor do we require any specific depth/level of decomposition. Our approach works well with a variety of document structures. Note that the document tree consists of internal nodes that represent structure, and all document content resides at leaf nodes.

### Distributing the Document Tree

We begin by assuming that all users have a copy of the shared document on their local, distributed editing systems. Each user/peer within the CES maintains a list of all other peers within the CES, and initially the CES contains no users. As discussed in our previous work [10], the document tree maintains locks about which users own which sections of the document; further, these locks are maximized using our previously-defined algorithms such that a user will always own the largest sub-tree that does not conflict with any other user's sub-tree. Whenever a user enters or leaves the CES (or moves from one section to another), locks are promoted or demoted.

Our previous work [10] employed a centralized approach to maintaining the document tree; we now adopt a peer-to-peer approach in distributing the document tree to avoid the bottleneck of the central server. When a user  $U_n$  is viewing/editing a section  $S_i$  of the document, that portion of the document tree representing  $S_i$  is held by  $U_n$  and the CCI-correct "most current" version of  $S_i$  is cached by  $U_n$ .

### Divergence and Convergence

Three different types of editing may occur within a CES: synchronous, asynchronous, and multi-synchronous [9]. We refine the notion of synchronous editing to include only those edits that involve two or more users within the same section of the document at the same time. Consequently, we may define asynchronous editing to include the traditional view of CES between multiple users at different times, but also add that editing made at the same time but within distinct/different sections of the document is also asynchronous.

As users edit a shared document, they may edit the same sections of the document (wherein conflicts occur and/or are highly likely); additionally, users may also edit disparate/distinct sections of the document (such that conflicts are not possible/present).

Concurrent operations/edits among multiple users within the same section of a document may be in conflict ( $\otimes$  as defined in [13]). Consequently, changes must be communicated immediately and methods such as OT must be employed to ensure that peers within the system are notified and the CCI properties [14][15] are ensured.

If we view editing disparate sections of a

document as compatible ( $\odot$  as defined in [13]) such operations may be executed in parallel and no immediate interaction among the edits is necessary; this is to say that these operations may be locally cached and sent later to the peers within the CES to achieve convergence, i.e. delaying the convergence.

We propose viewing the editing process as entirely multi-synchronous wherein changes are made that result in divergent local copies of the shared document; these changes must then be communicated to and incorporated later among peers to reestablish consistency.

Figure 1 demonstrates such a situation. White-rooted trees denote non-locked (no users present) sections; black-rooted trees denote locked (or multiple writers/users employing OT) sections cached locally; grey-rooted trees denote contention exists further down in the document tree (i.e., black-rooted trees must be present lower in the tree) [10].

We refer to sections of a document based upon their spatial relationships – denoting disparate sections based upon locality. Our approach is generalizable to define disparate elements of a document based upon some other, non-spatial criterion; for example, layers within a CAD document or drawing objects of different structure/types could be viewed as distinct and disparate regardless of the physical locality of the entities within the same locality of the document. Thus we use the term "disparate" to denote any non-conflicting section.

Initially (1), no users are in the document. At (2), user  $U_1$  enters the document, and since there is no contention,  $U_1$  edits the entire document without conflict from any other users. At (3), another user,  $U_2$ , enters and desires to edit a section of the document within the section rooted at node  $x$ ;  $U_1$ 's lock is demoted to node  $w$  (demote towards section



Figure 1: Distributing the Document Tree State among Peers

that  $U_1$  is currently active), and any changes made to the sections rooted at  $x$  are communicated to  $U_2$ . In (4), another user,  $U_3$ , enters and desires to edit a section of the document represented by the sub-tree rooted at  $z$ .  $U_1$ 's lock is demoted to node  $y$ , and any changes made to the sections represented by the tree rooted at  $z$  are communicated to user  $U_3$ . At this point,  $U_1$  contains the current version of the sections of the document rooted at  $y$ ,  $U_2$  contains the current version of the sections of the document rooted at  $x$ , and  $U_3$  contains the current version of the sections of the document rooted at  $z$ . If changes are made by  $U_1$ ,  $U_2$ , or  $U_3$  at this point, those changes are not sent to any other user since they are in disparate sections.

If  $U_1$  leaves the CES or moves to another section,  $U_3$ 's lock on  $z$  should be promoted to  $w$ , and changes made to the sections represented by  $y$  should be sent to  $U_3$ . We know this replay of changes to  $U_3$ 's copy of sections rooted at  $y$  is non-conflicting with any changes made to sections rooted at  $z$  because  $y$  and  $z$  are disparate. As a result, the history buffer rooted at  $w$  may be defined as the merging of the history buffers of  $y$  and  $z$  incurring little computational cost (it would be linear with respect to the size of the history buffers of  $y$  and  $z$ ).

#### **Caching Changes and the History Buffer**

Since all edits to a section are cached by the user making the edits, we maintain the history buffer of all edits to the section within the node of the tree representing the section. If the lock associated with this section is demoted (as the result of another user entering the section – as shown in Figure 1,  $2 \rightarrow 3$  and  $3 \rightarrow 4$ ), then this history buffer is sent to the new/entering user, and the changes can be replayed on the new user's local copy of the section. If the lock associated with this section is promoted (due to a user leaving/moving and removing conflict – as shown in Figure 1,  $4 \rightarrow 5$ ), then this history buffer is sent to the remaining user, and again the changes can be replayed on the remaining user's local copy of the section.

We employ the OT concept of the history buffer to maintain a list of all local changes made; additionally, if multiple readers/writers have adopted a sharing/OT approach within a section, these users can immediately receive changes from each other and employ standard OT techniques to ensure CCI. If the choice has been made to grant an exclusive write lock to one user within the section, then this history buffer maintains a differential (change log) for this version of the section's contents such that when promotion and/or demotion is necessary, changes can easily be relayed to other peers and replayed.

Users may be granted exclusive access to a portion of the document via our dynamic algorithm, but if the users choose to allow multiple writers within the same section of the document, these shared subsections can employ any modern OT technique ([3] and [8] as examples) locally within the subsection without interacting with users or document content outside of the subsection.

#### **ENSURING CCI VIA OT AND DYNAMIC LOCKS**

When there are two or more readers/writers within a single atomic section ( $S_i$ ) of the document, our system adopts a strict OT approach for all changes within  $S_i$ . Changes within  $S_i$  need only be sent to users within  $S_i$  (i.e., using selective multicast [7]). As a result, we achieve a reduction in communication cost proportional to how dispersed the users are (i.e., if the number of users within  $S_i$  is relatively small to the number of total users within the system, there is a significant reduction in communication cost). Of course, when other users enter  $S_i$  (or all users within  $S_i$  leave the CES), the changes made to  $S_i$  must be communicated. Such a delayed consistency approach retains good local response time without incurring unnecessary real-time communication costs to update users that are not immediately interested in such changes [11]. By combining multi-granular OT and dynamic, pessimistic locking, we enable responsive, real-time editing and also enable exclusive write access with user input deciding which policies to adopt; further, we can offer users the choice of whether edits should be accepted by other users (via voting protocols) when converging local changes among other users within the CES [2].

#### **Communication and Computation Costs**

As discussed in [17] and [10], it is not cost effective to immediately communicate individual key-stroke (character) level edit actions to other users within the CES; rather, such changes can be cached locally and sent as one edit action when possible to minimize the communication overhead of the network packets.

We agree with [1] that conflicts are a “naturally-arising side effect of the collaborative process” and “will occur simply because of the semantics of multi-user applications.” Further we agree with [4] that “temporary inconsistencies are necessary to achieve good performance” within collaborative editing systems. Thus at various points in time, the copies of the document are not consistent, but the distributed, managed copy of the document in its entirety is correct and preserves user intention; further, we record ownership and change history sufficient to recreate the entire document as needed (i.e., when a user wishes to view any specific section). These changes will be communicated and replayed among local copies as the users move about and view new sections, and changes can also be sent among the users (moving changes up the tree – minimizing communication costs) at specified intervals if desired [11][12]. Selective multicast is employed to improve communication cost [7].

When a user  $U_n$  enters the CES or moves to a new section  $S_i$  within the CES, he is notified of any changes that have been made to  $S_i$  (i.e., the state of  $S_i$  for  $U_n$  must be made current relative to all other users within  $S_i$ ). Because the tree rooted at  $S_i$  is maintained by the current owner  $U_m$  (and thus  $U_m$  holds the current/correct copy of  $S_i$ ), the new user  $U_n$  is guaranteed to correctly receive changes made to  $S_i$ .

## CONCLUSION AND FUTURE WORK

We have established our data structure to maintain the distributed document tree among readers and writers, presented dynamic algorithms to maintain locks and integrate OT techniques, and defined cache policies to minimize communication costs among participants in the CES. Additionally, our preliminary simulation results demonstrate that our approach of integrating dynamic locking and OT techniques can reduce computation and communication costs while maintaining responsiveness.

We are currently examining open-source repositories' change logs and performing data mining to detect patterns in how users edit software code; additionally, we are examining the structure of thousands of software code documents to detect patterns in document structure. From these results, we plan to enhance our simulations with more authentic models of clients' editing behavior and document structure. Additionally, we plan to further examine how various cache policies (varying the delay in updating among clients) and how adjusting the depth of promotion/demotion of locking effect computation and communication costs.

## REFERENCES

1. Edwards, W. K. Flexible Conflict Detection and Management In Collaborative Applications, Proceedings of the 10<sup>th</sup> ACM Symposium on User Interface Software and Technology (UIST'97). Banff, Alberta, Canada. October 14-17, 1997.
2. Greenberg, S. and Marwood, D., Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface, ACM Conferences on Computer Supported Cooperative Work, ACM Press, pp. 207-217, Nov. 1994.
3. Gu, N, Yang, J, and Zhang, Q. Consistency Maintenance Based on the Mark and Retrace Technique in Groupware Systems, *GROUP'05*, ACM Press, pp. 264-273, Sanibel Island, FL, Nov. 6-9 2005.
4. Handley, M. and Crowcroft, J. Network Text Editor (NTE): A scalable text editor for the Mbone, Proceedings of ACM SIGCOMM'97, pp. 197-208, Canne France, Aug 1997.
5. Ignat, C-L., and Norrie, M. C., Flexible Merging of Hierarchical Documents, In IEEE Distributed Systems online, ISSN 1541-4922. Proceedings of the Seventh International Workshop on Collaborative Editing, *GROUP'05*, Sanibel Island, Florida, USA, November, 2005
6. Ignat, C., and Norrie, M. Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems, In IEEE Distributed Systems online, ISSN 1541-4922. Proceedings of the Fourth International Workshop on Collaborative Editing, Computer Supported Cooperative Work (CSCW 2002), New Orleans, Louisiana, November 2002.
7. Li, D., Zhou, L., and Muntz, R.R., A New Paradigm of User Intention Preservation in Realtime Coollaborative Editing Systems, In Proceedings of the Seventh International Conference on Parallel and Distributed Systems, pp. 401-408, Iwate, Japan, July, 2000.
8. Li, R., and Li, D., A Landmark-Based Transformation Approach to Concurrency Control in Group Editors, *GROUP'05*, ACM Press, pp. 284-293, Sanibel Island, FL, Nov. 6-9 2005.
9. Molli, P., Skaf-Molli, H., Oster, S., and Jourdain, S. Sams: Synchronous, asynchronous, multi-synchronous environments, The Seventh International Conference on CSCW in Design, Rio de Janeiro, Brazil, September 2002.
10. Preston, J. A., and Prasad, S. K. Dynamic Locking of Varying Granularity in Generalized Document Trees to Maximize Concurrency and Minimize Communication in Synchronous CES, The 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing, Atlanta, GA USA, Nov. 17-20, 2006, under review.
11. Qin, X. Delayed Consistency Model for Distributed Interactive Systems with Real-time Continuous Media, *Journal of Software*, Vol.13, No.6, pp.1029-1039, June, 2002, China.
12. Qin, X., and Sun, C. Recovery Support for Internet-based Real-Time Collaborative Editing Systems, Proc. International Conference on Computer Networks and Mobile Computing, Oct. 2001.
13. Sun, C. and Chen, D. A Multi-version Approach to Conflict Resolution in Distributed Groupware Systems, Proceedings of thr 20<sup>th</sup> IEEE International Conference on Distributed Computing Systems, pp. 316-325, April 10-14, 2000.
14. Sun, C., Jia, X., Zhang, Y., and Yang, Y. A Generic Operational Transformation Scheme for Consistency Maintenance in Real-time Cooperative Editing Systems, In Proceedings of International ACM SIGGROUP Conference on Supporting Group Work, pp. 425-434, Phoenix, Arizona, USA, Nov. 16-19, 1997.
15. Sun, C., Jai, X., Zhang, Y, Yang, Y., and Chen D. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems, *ACM Transactions on Computer-human Interaction*, Vol. 5, No. 1, pp. 63-108, March 1998.
16. Sun, C., and Sosič, R. Optional locking integrated with operational transformation in distributed real-time group editors, In Proc. of The 18th ACM Symposium on Principles of Distributed Computing, pp.43-52, Atlanta, USA, May 4-6, 1999.
17. Yang, Y., Sun, C., Zhang, Y, and Jia, X., Real-Time Cooperative Editing on the Internet, *IEEE Internet Computing*, pp. 18-25, May/June, 2000.