

P2P Document Tree Management in a Real-Time Collaborative Editing System

Jon A Preston and Sushil K Prasad

Department of Computer Science
Georgia State University
Atlanta, GA

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

Abstract. This paper presents our work in combining peer-to-peer dynamic tree management with hierarchical Operational Transformation (OT) over document trees to achieve low computational and communication costs. We discuss our approach in storing the document tree in a peer-to-peer, distributed manner and maintaining convergence, causality preservation, and intention preservation (CCI) via a peer-to-peer caching system. Because changes are sent to other users within the system only as needed (and cached when possible), our approach minimizes communication costs among multiple readers and writers. Our algorithms balance the traffic and computational load among peers. They ensure that users always have the most current/correct copy of the section(s) of the document which they are viewing. Our approach outperforms existing OT techniques that broadcast messages and compute OT for each operation at all peers. This paper presents our algorithms and simulation results demonstrating the efficiencies and load balancing among peers within the system.

Keywords: Concurrent P2P tree, Load Balancing, Lazy updates, Communication efficiency

1. Introduction

The field of Computer Supported Collaborative Work (CSCW) and the subfield of Real-time Collaborative Editing Systems (RTCES) seek to achieve the goal of providing synchronous (real-time) access to a shared document. RTCES and consistency maintenance within distributed, synchronous RTCES are active research areas within the field of CSCW [18]. Most notable are algorithms [3][8] that work to achieve a high level of concurrent access through optimistic concurrency control and Operational Transformation (OT) to merge changes on other users' copies while ensuring consistency, causality-preservation, and intention-preservation – the widely accepted CCI model [17][18]. There exists a significant opportunity to reduce the computation and communication costs associated with OT [13].

We revisit the idea that locking offers RTCES research opportunities. 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 RTCES research community has adopted OT or similar approaches to maintain real-time, high responsiveness while striving to maintain the CCI model. Unfortunately, adopting an existing OT approach is costly in that all changes/operations must be broadcast to all users, and each of these users must “replay” these incoming

operations locally. This is costly with respect to communication and computation (since OT algorithms employ history buffers of previous operations which must be maintained and potentially modified).

Motivated by the idea that locking and OT serve complimentary roles [19], we incorporate both techniques such that users are able to make changes locally within locked sections of a document (and cache these changes) while adopting OT when many users make changes to a shared section of the document (multicasting these changes to only those users sharing the section). Whether a section is locked or shared is a user-defined policy that can change depending upon the scenario and on a per-section basis. By dynamically managing lock granularity, we maximize concurrent access among writers and avoid performing OT globally over the entire document. OT can be applied locally to a subsection of the entire document to reduce communication and computation costs.

Our approach employs a relaxed consistency model in which not all users within the CES have the most current copy of the entire document – rather, all users have the most current copy of the sections 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 heretofore enforced in OT-based systems, we are able to reduce communication and computation costs.

This research extends our previous work on centralized document trees [10]-[13] by distributing the document management among all peers within the CES and allowing the cached changes (history buffers) to be applied at an arbitrary level in the hierarchical document tree. These P2P algorithms for managing document sections and distributing existing OT algorithms are complimentary, superior to the current best practices of existing OT algorithms over linear document representations, and significantly reduce the computational and communication costs.

The remainder of this paper is structured as follows. We begin with a discussion of our hierarchical, tree-based view of the document, and then provide an overview of the data structures used to manage the document among the peers in Section 2. Next, we discuss our peer-to-peer algorithms to manage ownership of the sections of the document in Section 3. Section 4 presents a discussion of load balancing and fault tolerance, and Section 5 presents our simulation results validating our approach. We present related work in Section 6 and conclusions in Section 7.

2. Maintaining Exclusive Access

To avoid blocking users from editing, we use a tree-based view of the document and allow for locking at various levels within the tree such that users manage/lock local portions of the document. When a user desires a section for writing, we dynamically adjust lock granularity via demotion of the lock down in the document tree until the conflict among other users is resolved. When a user leaves a section of the document and makes it available to other users, conflict among users is potentially reduced; as a result, our approach automatically promotes the lock to a higher level within the document tree – maximizing the amount of the document owned for the remaining user [10].

2.1. Representing a Document via a Tree

By viewing the shared document as a hierarchical structure we are able to better achieve context-specific consistency preservation, and we can reduce communication and computational costs. Based upon the semantic structure of the document, the document may be broken up into sections, subsections, etc. The document tree consists of internal nodes that represent structure, and all document content resides at leaf nodes.

Each portion of a document may consist of content and sub elements (sub sections). Because any section of a document may contain any number of text elements (paragraphs, sentences, etc.), and may contain any number of subsections, we employ algorithms for inserting and removing locks from the collaborative space to work within an n-ary tree data structure that is representative of a shared document [10].

It is important to note that structure of the tree is defined within the document itself and does not depend upon any voting schemes or user involvement. For example, the structure of this paper is defined by the sections, subsections, paragraphs, sentences, etc.; the structure of a CAD file is defined by layers, objects, etc. Our approach does allow for users to establish at what granularity the leaf nodes should be defined, but this could also be defined automatically (defaults depending upon the type of document in use). Once established, the tree is utilized to manage ownership of subsections within the document. Rather than locking the entire document, lock granularity is adjustable, ranging from the entire document (ownership marked at the root of the tree) to an atomic level (ownership marked at a leaf node in the tree). The size of a subsection is not specified within our algorithms, thus it is scalable to accommodate the semantic structure of the document being edited.

2.2. Maintaining Node Coloring, Grey Count, and Black Siblings

To enable efficient management of which user owns/manages each section within the document, we utilize a tree coloring scheme. Each node in the document tree maintains a color (white, black, or grey) to denote whether it is available, currently being written to by another user, or if two or more users are editing sub-trees, respectively. Ownership (black coloring) of a vertex v by user U implies that U owns/manages v and the sub-tree rooted at v . We allow multiple users to concurrently write to the section denoted by a node and employ an OT consistency maintenance algorithm among the writers; thus a black node may have multiple owners/managers. Each node n in the tree maintains a numeric value that denotes how many users are managing nodes in the sub-trees of n . This count is defined as the *grey-count* of the node n . This grey count value is useful in efficiently determining if the node can be colored white or grey when a user leaves or moves and promotion is possible (as explained later).

A grey node v maintains references to the node's children (sub-trees); additionally, if there exists at least one black child node of v , then v also maintains a reference to the first black child node. The black child nodes of v (b_1, b_2, \dots, b_k , where

k = number of black child nodes of v) are linked together using a doubly-linked list. These sibling references and the reference to the first black child of a grey node are used for fast ($O(1)$) location of which node may be promoted when all but one user has left the sub-tree (as explained in Section 3.2).

When we say that user U owns a node v , we claim that this user contains the most current cached copy of the document's content represented by the node v . When we say that a set of users $\{U_1, U_2, \dots, U_n\}$ own a node v , we claim that these users all maintain the most current cached copy of the document's content represented by node v and are employing a localized OT algorithm to maintain consistency on the content of v . Thus the total correct, up-to-date copy of the document is distributed among the peers and can be constructed as needed by requesting the cached sections from the peers.

2.3. Lock Granularity, Localized OT, and Caching Local Changes

It is advantageous to maintain a lock on the largest sub-tree that is permissible; by maximizing the sub-tree that any user owns, we minimize the communication costs of the system by utilizing caching. For example, if a user U_i owns a section of the tree, then all changes to that section can be stored locally in the user's cache and do not have to be transmitted to other users until they desire to enter (read or write) that section.

Each node may employ a sharing policy to either allow or disallow multiple writers. If exclusivity is adopted at a node n (such that we allow only one writer), a lock on a sub-tree rooted at node n is permissible for user U_i so long as no other user has a lock on any node within the tree rooted at node n . If another user U_j enters the system and requests a section of the document, then the section of the tree owned by user U_i is reduced to accommodate the insertion of user U_j (assuming n is not an atomic/leaf node and demotion is possible).

Alternatively, if a multiple-writer policy is adopted at a node n , then multiple users may write to n and maintain consistency via OT. In this case, if n is owned by a single user U_i and another user U_j desires to enter n , then both U_i and U_j own n and OT is employed to maintain consistency. Any changes made by U_i to n before U_j 's arrival are transmitted to U_j upon arrival.

3. P2P Algorithms for Distributed Document Management

We agree with Edwards [2] 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 Handley and Crowcroft [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. 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 [14][15]. Selective multicast is employed to improve communication cost [7].

To minimize communication costs, locating which peer is currently managing a section of the document (currently managing a node in the document tree) may be done quickly via a peer location approach such as Chord [16]. Further, whenever a history buffer (Δx) for a node x is being transmitted, reduction of Δx to $\Delta x'$ can also decrease the communication cost.

Given the structure of the document tree and the distributed nature of the tree management, when a user u manages a section denoted by node n , then all changes made to the content of the sections rooted at n are cached locally on u . Thus any structural change to the document tree (such as combining two sections, splitting a section into two sections, or deleting a section) can also be cached locally on u . If a set of users are sharing n and employing OT to maintain consistency of n , then any structural changes within the sub-tree rooted at n will be maintained via the OT algorithm on n among the users sharing n .

As a result, the algorithms specific to handling the placement and management of users within the document tree are the focus of this research. The two algorithms we have developed are USERENTER and USEREXIT. We present each below with a detailed analysis of correctness and efficiency. As in our previous, centralized algorithms [10]-[13], we avoid deadlock among peers by employing handshake locks on parent/child nodes and by always moving downward through the tree.

3.1. User Entering a Section

When a user desires to write to a section s of the shared document, the user must enter the section of the document tree that represents the desired section s . This operation is performed by the USERENTER algorithm that works from top-to-bottom by examining nodes in the path from the root to the desired node. The correct path is determined by first querying the peer who manages the root, and then descending further down by following peers' references to other peers. The coloring of the nodes along the path denote the availability of the sections (see Section 2.2). If a white node is reached, then the node becomes managed by the user (and marked black denoting the ownership). If a grey node is reached, the algorithm proceeds further down the path. If a black node is reached, then the user's entry to the section depends upon the sharing policy adopted at the black node. If the sharing policy is set to exclusive write, then the entry fails (i.e., the user cannot enter the desired section of the document); if the sharing policy is to adopt OT (allowing multiple writers at this node), then the requesting user is added to the set of concurrent writers for the section; finally, we can adopt a *demote* policy. Demotion works by moving the management of the existing user down the tree hierarchy until the conflict among the users is resolved.

The most complicated case of the USERENTER algorithm is when demotion occurs. Figure 1 demonstrates the demotion of U_1 from the section v down to the subsections denoted by $\{w_1, \dots, w_n\}$ and the injection of U_2 at the section denoted by x . Any history buffer at U_1 to x (denoted by Δx) must be passed to U_2 . At this point,

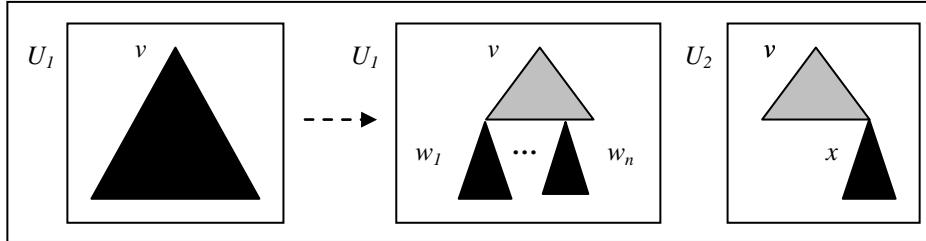


Fig 1. Adding U_2 by reducing and distributing U_1 's management of v

U_1 contains the most current copy of the sections $\{w_1, \dots, w_n\}$, and U_2 contains the most current copy of section x .

Correctness: As there are only three colorings for any node (white, black, and grey), the USERENTER algorithm handles all cases. In the case where n is colored white (1), there are no other users in n , so u obtains n . In the case where n is colored black (2), we have three cases on resolving the entry of u to n : (2.1) OT is adopted at n , so u is added to the user set of n ; (2.2) n is not shareable in which case the entry of u to n fails and we must decrement the grey count of the nodes along the path from the root to n to keep the grey count correct (since the node n was not granted to u); and (2.3) when demotion occurs and n is colored grey. In the more complicated case of 2.3, the current manager of n (v) is demoted along the path such that the original request of v (that led to v 's management of n) is still fulfilled. This either resolves the conflict (2.3.4) or we repeat the invocation of the algorithm one level down in the tree closer to w (2.3.5). In the case where n is colored grey (3), we increase the grey count (optimistically assuming u 's request for w will be successful) and repeat the invocation of the algorithm one level down in the tree closer to w .

Performance Analysis: Since the USERENTER algorithm traverses from the root down to a leaf (or stops earlier if a white or black node is reached), this algorithm must traverse $O(h)$ nodes, where h equals the height of the document tree. The work involved at each node is $O(1)$ since the work in processing an individual node involves updating references/pointers, coloring, and grey count (integer) values. It is possible upon a failure (step 2.2) that the USEREXIT function will be invoked, but this USEREXIT (as discussed below) runs in $O(h)$, thus it is not asymptotically greater than the existing $O(h)$ work for the USERENTER algorithm. Thus the computation cost for the USERENTER algorithm is $O(h)$. As the algorithm traverses down the tree, peers that manage each of the nodes along the path must handle the request; thus as many as $O(h)$ peers must be involved in resolving the request – and this incurs $O(h)$ messages. Communication also occurs when the lock is granted and the history buffer is communicated to the requesting user for a cost of $O(b)$ where b is the size of the single history buffer communicated. In the case where we are adding u to the manager set of n (2.1), this requires $O(n)$ messages where n is the number of users in the OT set of n (since all users in the set must be notified of the user entering the set). Thus the total communication cost for USERENTER is $O(n + b + h)$.

USERENTER(*n*, *w*, *u*)

Input: node *n* representing a section of the document tree, node *w* representing the desired node, and a user *u* that is the user/peer who wants to write to *w*.

Output: a reference to the node *x* that *u* owns such that *x* is a root of a sub-tree that contains *w* as a leaf node (return null in the case where it is not possible to fulfill *u*'s request to enter *w*)

Assertions: *n* is the root of a sub-tree that has *w* as a leaf node (such that *n* handles the request of user *u* to be able to write to *w*), *v* is the current manager of *n*

```

1.    if n.color = WHITE
        SetManager(n, u, w) // n.color BLACK and n.originalRequest = w
        LinkToSiblings(n)
        Return n
2.    else if n.color = BLACK
2.1.   if n.policy = OT
            Communicate( $\Delta n$ , u)
            AddToManagers(n, u, w)
            Return n
2.2.   else if n.policy = EXCLUSIVE or n.IsLeaf // demotion not possible
            USEREXIT(root, w, u) // correct artificially-inflated grey counts
            Return null // failed entry due to lack of sharing on n
2.3.   else // demotion must occur
2.3.1.    n.color = GREY
2.3.2.    n.greyCount = 2
2.3.3.    Demote v down to NextInPath(n, n.OriginalRequest)
2.3.4.    if NextInPath(n, n.OriginalRequest) != NextInPath(n, w)
            Communicate( $\Delta n$ , u)
            Return NextInPath(n, w)
2.3.5.    else return USERENTER(NextInPath(n, w), w, u)
3.    else // color is GREY
        n.greyCount++
        return USERENTER(NextInPath(n, w), w, u)

```

3.2. User Leaving a Section

When a user desires to exit to a section *s* of the shared document, the node in the document tree that represents the *s* must remove the user from its list of managers/writers. This operation is performed by the USEREXIT algorithm that works from top-to-bottom by examining nodes in the path from the root to the node to release. As in the USERENTER algorithm, the correct path is determined by first querying the peer who manages the root, and then descending further down by following peers' references to other peers. Again, the coloring of the nodes along the path indicate how to handle the request to leave. If a black node is reached, then we remove the user from the set of users managing the node and inform all remaining users within the section that the user has left. If a grey node is reached, there are three cases: the grey count is decremented and the algorithm proceeds down the tree to the node next in the path to the desired node; the grey count is decremented to 0 which means the exit is complete as all users have left the section represented by this sub-

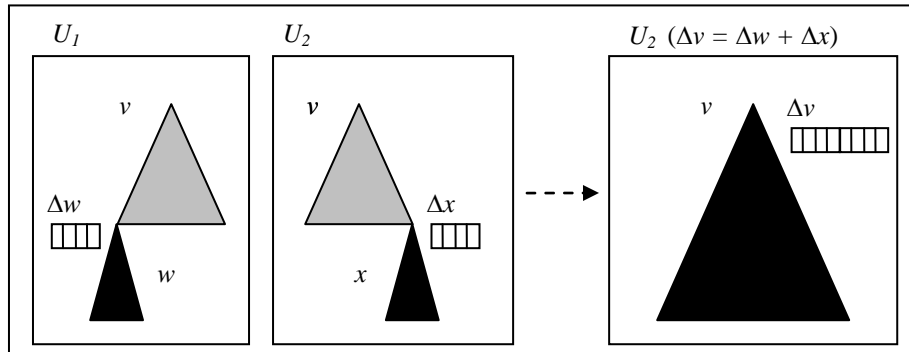


Fig 2. Removing U_1 by promoting U_2 & communicating Δw to U_2

tree; the third case is when promotion is possible because the grey count is going from 2 to 1 (denoting there will only be one user remaining after the exit). Promotion works by finding the remaining user (the user not leaving) and moving their management up to the current node being examined. This remaining user can assume ownership of a larger portion of the document (and maximize the caching).

The most complicated case of the USEREXIT algorithm is when promotion occurs. Figure 2 demonstrates the promotion of U_2 to v when U_1 leaves w . As a result, Δw (the history buffer of w) is communicated from U_1 to U_2 . At U_2 , x is current since U_2 already managed it, and w is now current because Δw has been “replayed” at U_2 . Thus U_2 contains a proper and complete, up-to-date version of v since v is defined by w and x (i.e., v is current because $v = w + x$ and $\Delta v = \Delta w + \Delta x$). When Δw is communicated to U_2 , U_2 may elect to incorporate Δw into its copy of w , or U_1 's changes to w may be rejected. This acceptance or rejection of changes by other users could be done automatically by the system based upon embedded rules or done explicitly by users as prompted by the system.

Correctness: The USEREXIT algorithm handles only two cases for a node's coloring: black and grey. The algorithm does not handle the case where the color is white because a white coloring denotes the node n is not managed by any user, thus no user can exit n . In the case where n is colored black (1), the user u is removed from the user set managing n and removed from the sibling list. Promotion is not possible or it would have occurred earlier in the path from the root to n (as discussed below). In the case where n is colored grey (2), the grey count is decremented to denote u is leaving the sub-tree rooted at n (2.1). We then have three cases on resolving the exit of u from n : (2.2) is the case when a promotion occurs since only one user remains in the sub-tree n , so we promote the only remaining user (other than u) to n and communicate the history buffer of the node u is managing to the remaining user; (2.3) n no longer has any users remaining in it, but this coloring to white was temporarily delayed due to a previous, concurrent invocation of USERENTER that failed; and (2.4) when we repeat the invocation of the algorithm one level down in the tree closer to w

Performance Analysis: USEREXIT traverses from the root down to a leaf (or stops earlier if a grey or black node is reached), this algorithm must traverse $O(h)$ nodes, where h equals the height of the document tree. The work involved at each node is $O(1)$ since the work in processing an individual node involves updating references, color, and grey count (integer) values. Upon promotion, the FINDELIGIBLEPROMOTION function must be called, but it continues the traversal down the tree from the point where the promotion may occur, thus its work is $O(h)$. Thus the overall computation cost for the USEREXIT algorithm is $O(h)$. As the algorithm traverses down the tree, peers that manage each of the nodes along the path must handle the request; thus as many as $O(h)$ peers must be involved in resolving the request – and this incurs $O(h)$ messages. Communication also occurs when the proper peer is located to fulfill the request; there are two cases when fulfilling the request: either promotion occurs or it does not. In the case of promotion (Step 2.2), one history buffer is communicated to the user that is promoted, thus the communication cost is $O(b)$ where b is the size of the history buffer. In the case when promotion does not occur (Step 1), the requesting user must be removed from managing the node. This requires $O(n)$ messages where n is the number of users in the OT set of n (since all users in the set must be notified of the user leaving the set). Thus the total communication cost in USEREXIT is $O(n + b + h)$.

USEREXIT(n, w, u)

Input: node n representing a section of the document tree, node w representing the node no longer desired, and a user u that wants to leave w .

Output: none

Assertions: n is the root of a sub-tree that has w as a leaf node (such that n is fulfilling the request of user u to be able to leave w)

1. if $n.color = BLACK$
 RemoveFromManagers(n, u) // possibly $n.color$ WHITE (if no more managers)
 UnlinkFromSiblings(n)
2. else if $n.color = GREY$
 - 2.1. $n.greyCount--$
 - 2.2. if $n.greyCount = 1$
 $a = FindEligiblePromotion(n, w)$
 SetManager($n, a.manager, a.originalRequest$) // promote a
 $b = NextInPath(n, w)$
 Communicate($\Delta b, a.manager$)
 - 2.3. else if $n.greyCount = 0$ // previous failed UserEnter and two users have exited n
 RemoveFromManagers(n, u) // $n.color$ is now WHITE
 - 2.4. else *UserExit(NextInPath(n, w), w, u)*

4. Load Balancing & Fault Tolerance in the P2P Implementation

One motivation in developing our P2P dynamic locking algorithms was to distribute the work of lock management among the peers. Initially, it would seem that this work and communication is distributed uniformly among the peers, but the problem

remains that all messages must be processed from the root down as the grey counts must be modified from the root to the desired node to ensure proper promotion and demotion. Thus if a single peer is responsible for managing each node in the tree, some peer must maintain the root and will then become the bottleneck and have an increase in workload when processing the USERENTER and USEREXIT requests.

We achieve workload balancing by adopting a rotating management of the nodes in the tree. When an USERENTER operation is performed, the user begins management of the nodes along the path in the document tree visited in fulfilling the USERENTER operation. In this manner, we adopt a “most recently requested” policy in that all nodes n_i will be managed by the user who’s USERENTER request was fulfilled by passing through n_i (i.e., n_1, n_2, \dots, n_k is in the path from the root to n_k , where n_k is the desired node or the node at which the lock request is fulfilled). We note that when a USEREXIT operation is performed, this implies that the user is leaving a section and thus it is not advantageous to have the user begin management of nodes.

If such a “most recently requested” policy for lock management is adopted, then a single peer p must serve at most $O(n)$ consecutive lock management operations, where n is the number of peers in the collaboration. This is true because if a USERENTER request is handled, then the node acquires a new manager other than p . Only USEREXIT requests can be fulfilled and keep the same manager p , and there can be at most n consecutive USEREXIT request since any more would necessitate a lock request (i.e., a peer can’t release a lock it doesn’t have). The workload for a peer is proportional to the number of lock requests for the peer – thus more active peers in the RTCES will be responsible for handling more of the work in maintaining the document tree. And if the peers perform an equivalent number of lock requests, the workload of managing the nodes within the document tree is balanced since the amortized time a peer manages a node should be approximately equal to the amortized time the other peers manage the node.

The time a peer manages a node is proportional to the depth of the node in the document tree (since there are fewer paths that travel through a node at a greater depth than a node at a more shallow depth). Thus the root management should change more often than a near-leaf node. This is good because the workload of more shallow nodes in the tree (closer to the root) is more than the workload of deeper nodes. As a result, the workload in managing the distributed, P2P version of the document tree is balanced among the peers.

Another important property of our P2P approach is increasing the reliability and fault tolerance so that if one peer is dropped from the RTCES, the others can continue without any problems. We may increase the reliability and fault tolerance of the document tree by replicating the top portion of the tree among all peers (or a subset of peers). For reliability, we can adopt an n-way replication of sections of the document and consistency maintained via an OT policy. While this increases the communication cost (since all peers must perform OT to maintain consistency regarding the lock states among the replicated portion of document tree), this approach does overcome the single point of failure of a single manager for a document section. If this n-way replication is adopted, then the sections of the document would implement a queue to store the managers of the section of the document – thus the peer who had managed the replica the longest would rotate out

when a new peer's request arrived down the document tree (a LIFO approach to achieve load balancing).

5. Simulation and Results

To validate our P2P distributed document management approach, we implemented the model of the node and the USERENTER and USEREXIT algorithms. We modeled three different document trees containing 14, 28, and 56 leaves, respectively. We simulated concurrent users that were either in a reading or writing state; additionally, the users could move to a new section of the document (moving their cursor position), and this new section to which to move was randomly selected. A total of 96 simulation configurations were performed, varying among the three documents and increasing the number of users from 1 to 32.

The results of the 96 simulation runs are shown in Figure 3. Each column denotes a set of peers varying from 1 peer (in simulation runs 1-3) to 32 peers (in simulation runs 94-96). The workload is measured by how many USERENTER and USEREXIT requests were handled on a per-peer basis, thus each point plotted denotes how much work a single peer handled. Note that the y-axis is logarithmic to enable the variance among the peers within the columns to be visible.

If we adopt a first-come policy of node management, then as predicted, one (or a small few) peers are unfairly burdened with the bulk of the document management. Notice the high trend line showing the most burdened peer for each simulation run. When the "most-recent," balanced approach is adopted (see Section 4), the work is more fairly distributed among all peers. This is corroborated in that while the total work remains the same, the variance among the peers for any simulation run decreases when a balanced approach is adopted (note the increased clustering).

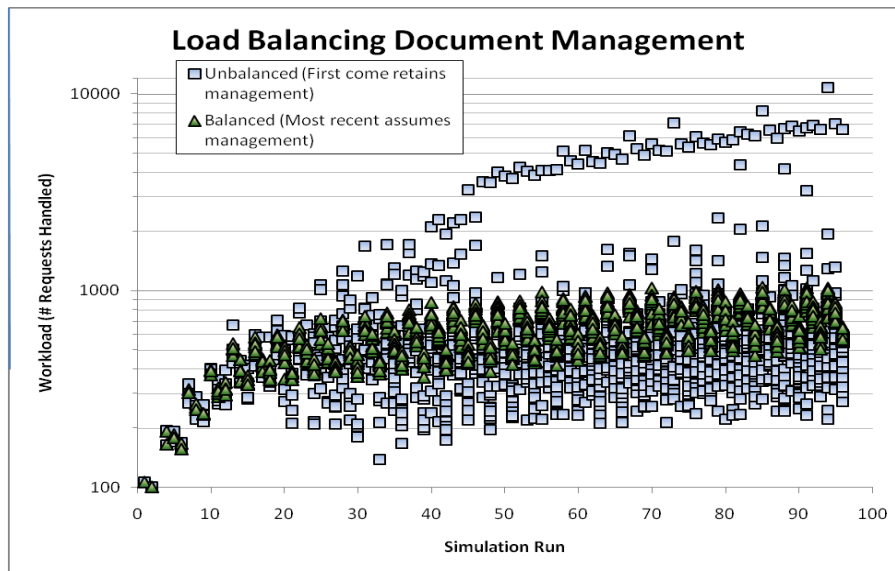


Fig 3: Balancing the Workload of Document Management among Peers

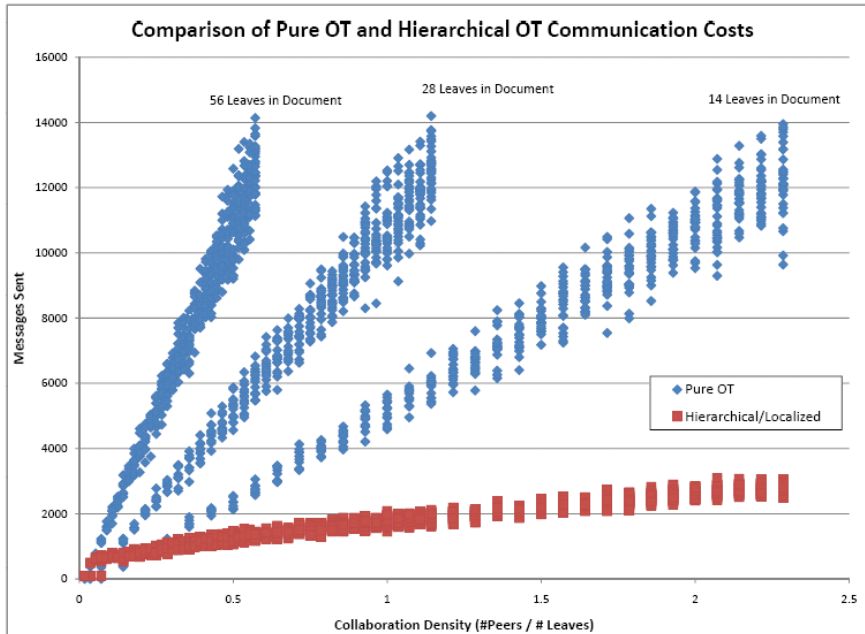


Fig 4: Pure OT vs. Hierarchical OT Communication Costs

observe that the total workload decreases when the document size increases. This is intuitive in that if we increase the document size while retaining the same number of peers, then the opportunity for caching increases under our distributed document management model.

Figure 4 shows how our hierarchical distributed document management approach can reduce the communication costs when compared to a pure OT approach. The ability to cache changes locally and localize OT to a subset of users sharing the same space within the document dramatically decreases the communication costs of the RTCES. We note that as the collaboration density (the average number of peers per section of the document) increases, the communication also increases; this is as expected since more messages will be sent to maintain consistency when more than one peer shares a section of the document.

It is important to note that the size of the document does not affect the workload in managing the collaboration – only the collaboration density affects the workload; thus our algorithms scale to large documents well.

6. Related Work

Traditionally, research within CES has viewed documents to be a linear sequence of data; consequently, OT and other techniques to ensure the CCI model [18] 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 [6][12]. 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 and intention preservation [6][19].

Others have examined utilizing varied-granularity locks within a shared document to achieve increased concurrent access and avoid the bottleneck traditionally associated with pessimistic, lock-based concurrency control [1]. [2] points out that requiring users to manage the maintenance of fine-grain locking is onerous and prohibitively costly, outweighing any benefit of such increased concurrency. The POEM and MACE systems utilize hierarchical locks at a sub-file level, but the locks must be defined a priori and are fixed in size [8]. In contrast, our algorithms automatically obtain the largest lock permissible and automatically adjust the size/portion of the document locked.

[6] incorporates the idea of varied-granularity into OT-based concurrency control algorithms by allowing the user to specify at what semantic level within the document OT merging should occur. This is similar to [1] and [5] that apply OT algorithms to SGML/HTML/XML. It is notable that our algorithms differ from [5] in that they do not block or require 2-phase locking.

Communication costs may be improved by employing OT at a greater-than-character level due to the fact that this approach would use more of the network packet size than a single character (thus avoiding wasteful single-character payload packets). As discussed in [10] and [20], 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 to minimize the communication overhead of the network packets.

7. Conclusion

We have presented peer-to-peer algorithms that dynamically manage ownership of a distributed document among peers efficiently, minimizing computation and communication costs. Additionally, we have presented a distributed version of OT algorithms that reduce computation and communication costs as compared to existing multicast-based OT algorithms. Both of these techniques are complimentary and improve the field of CSCW and RTCES.

Our empirical simulation results demonstrate that the work of managing the document and the document tree can be distributed fairly among the peers within the RTCES, and the “most-recent” rotating management policy removes the bottleneck of any single peer in managing the topmost nodes of the tree. Additionally, we have shown that we can achieve fault tolerance by replicating the top portion of the tree among a set of peers and maintain consistency via existing OT algorithms.

References

- [1] Davis, A. H., Sun, C., and Lu, J. “Generalizing Operational Transformation to the Standard General Markup Language”, Procs. CSCW 2002, New Orleans. Nov. 16-20. pp. 58-67.
- [2] Edwards, W. K. “Flexible Conflict Detection and Management In Collaborative Applications”, Procs. 10th ACM Symp. on User Interface Softw. and Tech. (UIST’97). Banff, Canada. Oct. 14-17, 1997.

- [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, Procs. ACM SIGCOMM97, pp. 197-208, Cannes, France, Aug 1997.
- [5] Helmer, S., Kanne, C-C., and Moerkotte, G., Evaluating Lock-based Protocols for Cooperation on XML Documents, SIGMOD Record, 2004.
- [6] Ignat, C-L., and Norrie, M. C., Flexible Merging of Hierarchical Documents, Procs of the Seventh Intl Workshop on Collaborative Editing, GROUP'05, Sanibel Island, Florida, Nov., 2005
- [7] Li, D., Zhou, L., and Muntz, R.R., A New Paradigm of User Intention Preservation in Realtime Coollaborative Editing Systems, In Procs. of the Seventh Intl. Conf. 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] Magnusson, B., "Fine-Grained Version Control in COOP/Orm", European Conference on Computer Supported Cooperative Work 1995, Workshop on Version Control in CSCW Applications, Stockholm, Sept. 1995.
- [10] Preston, J. A. and Prasad, S. K. "A Deadlock-Free Multi-Granular, Hierarchical Locking Scheme for Real-time Collaborative Editing", 7th Intl. Workshop on Collaborative Editing Systems. Sanibel Island, FL, 2005.
- [11] Preston, J. A. and Prasad, S. K. "Achieving CCI Efficiently by Combining OT and Dynamic Locking with Lazy Consistency in a Peer-to-Peer CES", 8th Intl. Workshop on Collaborative Editing Systems. Banff, Canada, 2006.
- [12] Preston, J. A. and Prasad, S. K. "An Efficient Synchronous Collaborative Editing System Employing Dynamic Locking of Varying Granularity in Generalized Document Trees", Procs. 2nd Intl. Conf. on Collaborative Computing: Networking, Appln. and Worksharing, Atlanta, Nov., 2006.
- [13] Preston, J. A, Xiaolin, H., and Prasad, S. K. "Simulation-based Architectural Design and Implementation of a Real-time Collaborative Editing System," Procs. 2007 DEVS Integrative Modeling and Simulation Symposium, Norfolk, VA, 2007.
- [14] Qin, X. Delayed Consistency Model for Distributed Interactive Systems with Real-time Continuous Media, Journal of Software, Vol.13, No.6, pp. 1029-39, June, 2002, China.
- [15] Qin, X., and Sun, C. Recovery Support for Internet-based Real-Time Collaborative Editing Systems, Proc. Intl. Conf. on Computer Networks and Mobile Computing , Oct. 2001.
- [16] Rao, V. N and Kumar, V. Concurrent Access of Priority Queues. IEEE Trans. on Comput.. Vol 37, No 12. pp. 1657-65. 1988.
- [17] Sun, C., Jia, X., Zhang, Y., and Yang, Y. A Generic Operational Transformation Scheme for Consistency Maintenance in Real-time Cooperative Editing Systems, In Procs. of Intl. ACM SIGGROUP Conf. on Supporting Group Work, pp. 425-434, Phoenix, Nov. ,1997.
- [18] 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 Trans. on Computer-human Interaction, Vol. 5, No. 1, pp. 63-108, March 1998.
- [19] Sun, C., and Sosič, R. Optional locking integrated with operational transformation in distributed real-time group editors, In Proc. of The 18th ACM Symp. on Principles of Distributed Computing, pp.43-52, Atlanta, May 4-6, 1999.
- [20] 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.