

Simulation-based Architectural Design and Implementation of a Real-time Collaborative Editing System

Jon A Preston, Xiaolin Hu, and Sushil K Prasad

Department of Computer Science

Georgia State University

Atlanta, GA USA

jon.preston@acm.org, xhu@cs.gsu.edu, sprasad@gsu.edu

Keywords: DEVS, real-time collaborative editing, OT, dynamic locking, communication costs

Abstract

Real-time collaborative editing systems allow multiple users to synchronously edit a shared document in a geographically-distributed environment. In order to maintain high responsiveness, a distributed copy model is used wherein each user maintains a local copy of the shared document; as a result, techniques such as Operation Transformation (OT) are employed to ensure consistency among the copies; but OT is costly with regard to computation and communication. We have previously developed a distributed architecture and associated algorithms that dynamically lock sections of a document such that users are able to retain a high level of responsiveness within the system while reducing the computation and communication costs. However, a key limitation has been locking out all contending users except one for the smallest indivisible subsection, resulting in scenarios with significant lock request failures. This study expands our work by examining how OT may be integrated into our dynamic locking algorithms such that all users are always able to edit their copy of the document while avoiding costly global messaging. We present an overview of our updated architecture and algorithms, show how they have been simulated using the DEVSJAVA package at the client and server, and then demonstrate the efficiencies achieved by our approach relative to existing OT algorithms. In scenarios featuring clustered editing, large document, and a large number of users, our system incurs up to 80% less communication cost than existing pure OT systems. Additionally, we discuss how our simulation design process has allowed us to first simulate both client and server and then begin progress to a functional implementation of both client and server technologies – better achieving an efficient implementation of our algorithms and ideas based upon our empirical simulation results.

1. INTRODUCTION

Real-Time Collaborative Editing Systems (RTCES) research is a subfield of Computer-Supported Collaborative Work (CSCW) that seeks to provide synchronous (real-time) access to a shared document. One of the goals of RTCES is to enable geographically distributed teams to work together, share ideas, collaboratively edit a shared document in real-time, and interact as closely and productively as a team of workers within the same room. Because of the need for high responsiveness in the client editing system, nearly all RTCES employ a distributed copy approach wherein each client maintains a copy of the shared document. As a consequence, work must be done to ensure these distributed copies remain synchronized – ensuring they maintain CCI: convergence, causality-preservation, and intention-preservation [1][2]. Section 2 contains the relevant background.

We revisit the idea that locking may be beneficial in RTCES to achieve CCI efficiently. Motivated by the idea that locking and OT serve complimentary roles [3], 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 when necessary (when many users make changes to a single section of the document and reconcile). We recognize that using locks reduces concurrent access, thus the CES community has focused on the optimistic approach that requires OT. In response, we have developed a set of algorithms that dynamically manage lock granularity and maximize concurrent access among writers [4][5][6]; as a result, we are able to avoid global computations and communication among all clients while providing for improved concurrent access.

In this study, we simulate a distributed RTCES that make use of our algorithms and measure the communication and computation costs given various client and document configurations. We compare our results to demonstrate our efficiency over existing OT-based approaches. We first model both the clients and the server to gather empirical evidence to support our approach. Our results indicate communication costs can be reduced by up to 80%; based upon these results, we are currently working to integrate the simulated models of the clients into a fully-functional server

running our algorithms to gather performance results that would not have been available using real-world users. The scalability of our approach is a significant contribution to the field in that no other RTCES has been tested with such a large number of clients.

The remainder of this paper is organized as follows: we give a background of the domain of RTCES, CCI, and OT in Section 2. Next, we discuss our simulation design process and the DEVS models in Section 3. In Section 4, we present our simulation results and comparative analysis. In Section 5 we discuss the work we are currently performing using the real server, and we provide conclusions and a discussion of future work in Section 6.

2. REAL-TIME COLLABORATIVE EDITING

Ensuring CCI is central to RTCES research, and there exist approaches that achieve consistency and causality-preservation. The most widely researched approach in achieving CCI centers around communicating all local changes to other clients within the RTCES and then performing transformations on these incoming operations to ensure they have a consistent effect relative to any operations that were enacted concurrently by the local client. This approach is defined as Operational Transformation (OT), and the need for it is illustrated in the time-space diagram shown in Figure 1. Three examples of the most recent OT approaches are discussed in [7][8] and [9].

While OT-based approaches are the most widely accepted approach to achieving consistency among distributed copies of a shared document with RTCES and there has been significant OT-based work done in solving some of the problems in achieving CCI, no OT approach is able to achieve intention preservation. Further, OT approaches broadcast all changes immediately to all users (incurring significant communication costs) and require significant memory and computation to maintain history

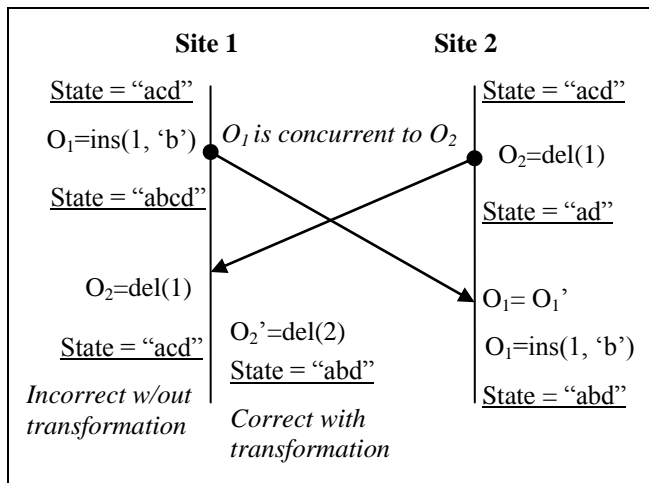


Figure 1 - Demonstrating the need for OT

buffers at each client. As a result, there exists a significant opportunity to reduce the computation and communication costs associated with OT. Recently, others have explored the idea of using tree hierarchies, decomposing the document into sections and more efficiently performing OT [10][11].

Employing locks avoids the need to perform OT and merge disparate copies of a shared document, but using locks reduces concurrent access; thus a lock-only approach is not sufficient for supporting RTCES. Rather, our approach views the shared document as a tree, dividing the document into sections and subsections as illustrated in Figure 2, and locks may be applied at a sub-document level to help avoid

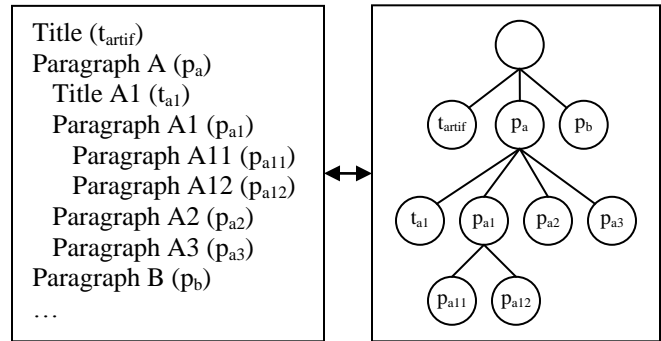


Figure 2 - Modeling a Document as a Tree

blocking other users. When desired (at a specified depth or on a per-user basis), OT may be employed within sub-trees. There is no limit to the depth of the tree, as its structure is dependent upon the semantic structure of the originating document. While we refer to text documents in this study, our approach is applicable to a variety of document types such as CAD and other object-based document types.

2.1. Messages within the RTCES

Within our system, locks are demoted and promoted as users request ownership of sections. Since all content resides at leaf nodes, our algorithms work from the root down to the desired leaf and utilize a coloring scheme to detect the sub-tree that may be granted to the requesting client such that the space the client owns is maximized. Further details of our algorithms may be found in [4], [5], and [6]. As the purpose of this simulation is to measure communication costs, we define messages passed among clients and the server as:

1. *Document Check-out (CO)* – the client would like to check out and become a reader of a document.
2. *Document Check-in (CI)* – the client is no longer interested in the document and releases it.
3. *Lock Request (LK)* – the client wants to write to a section of the document
4. *Unlock (ULK)* – the client has left the section and no longer needs the ability to write to it

In response to each of the above messages from a client to the server, the server may respond that the request succeeded or failed – for a total of eight (8) response types.

Further, since an existing client who owns a section of a document may have his lock promoted (moved up in the tree such that the client owns more of the document) or demoted (moved down in the tree such that the client owns less of the document), clients may also receive the following messages from the server indicating their new ownership status:

5. *Promotion (P)* – informs the user that he now owns more of the document that he previously owned.
6. *Demotion (D)* – informs the user that he now owns less of the document that he previously owned.

Additionally, messages must be passed to clients when a new user is added into the set of users writing to a section concurrently; these clients must perform OT among themselves to ensure CCI within the section of the document. Thus we have the following messages:

7. *OT Added (OTA)* – signals a user within a section that another user has been added to the section and future changes must be sent to this new user
8. *OT Deleted (OTD)* – signals a user within a section that a user has left the section and no longer needs to have changes sent to him
9. *OT Join (OTJ)* – tells the user requesting a lock that he has been granted write access to a section that is already using OT; this message contains a list of the existing users within the section so that the new user can send future changes to these users
10. *OT Modify (OTM)* – this message tells a client that the section has been modified and a local OT must be performed based upon the operation being communicated.

The simulations in this study incorporate each of these messages, as we discuss later in Sections 3 and 4.

3. DESIGN PROCESS AND MODELS

System test and performance evaluation are essential in a system development to ensure the system/algorithms under development will not cause major problems when deployed in the real field and used by real users. This is especially important for distributed systems, such as RTCES that has a large number of potential users. Unfortunately, the user-oriented nature of the system prohibits extensive testing and performance evaluation using real users. In this work, we follow a stepwise simulation-based design process to test/evaluate the system and algorithms under development. This stepwise design process is motivated by an earlier work [12] that develops a simulation-based design process to enable smooth transitions between different design stages. It aims to support systematic and cost-efficient testing and evaluation for the distributed collaborative editing systems concerned in this paper.

3.1. Simulation-based Software Architectural Design Process

The stepwise simulation-based design process includes three steps as shown in Figure 3. In the first step (a), both the server and clients are modeled as DEVS models; clients may have different profiles based on knowledge extracted from real user behavior extracted by analyzing change log files of document repositories. We apply a fast simulation approach wherein events advance the system clock and the simulation completes as fast as possible. At this stage, different configurations (such as varying the number of clients and/or client behavior patterns) can be easily setup, and multiple runs of the simulation may be quickly executed. A key advantage to this approach is that it is very flexible, and we are able to quickly get results without the need to fully implement a research server; this allows for testing and evaluation in the very early stages of the architectural design process. In the second step (b), the server is coded and fully implemented and run on a dedicated computer; simulated client models interact with the server through the network. The key advantage is that there is still flexibility for configuring the tests on the client side, such as having a large number of client models; this is especially cost efficient as no real users are involved and we can scale the tests beyond current RTCES testing user levels. In the last step (c), measurement data are collected where real users use the client editors to interact with the real server. At this stage, we are able to achieve high fidelity measurement of data because this consists of real users and the real server.

3.2. Modeling the Client and Server

In our previous DEVS Java simulation [13], we focused on decreasing the contention among shared documents by segmenting the documents into fixed subsections. In this work, we increase the complexity of how the lock proxy manages the subsections – using our more complex tree algorithms, but the overall structures of the simulation models remain consistent. The simulation models consist of a client machine, a network, and a server machine.

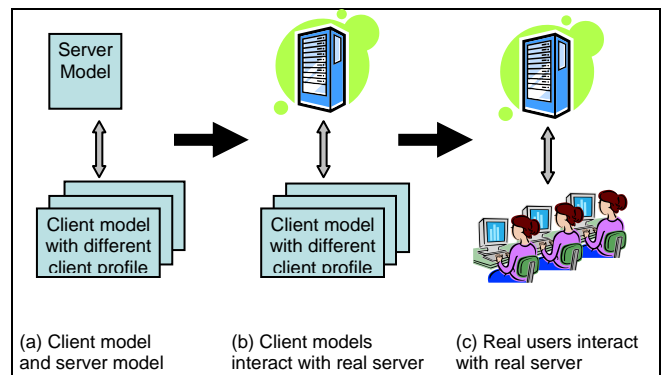


Figure 3 – Stepwise Simulation-based Software Design

The **client machine** is modeled to act as a state machine that begins outside of the document, may check out the document and becomes a reader, and then is either reading or writing to a specified section of the document [6]. When the client requests to write to a section, a lock request message is sent to the server and the server responds by notifying the client how much of the document it owns. The client is then free to move within the owned space and make changes, caching these changes locally. If the client receives a promotion or demotion message from the server, then it adjusts its ownership space accordingly and flushes its cache as needed. In contrast, if the client is sharing a section with other clients (via OT), then changes must be communicated immediately to the other clients.

The client editing behavior is determined as either *random* (the client will randomly move within different sections of the document) or *clustered* (the client's editing will be centered on a point within the document and the client will move within a small space around that point), and a *hybrid* that acts as a mix between the random and the clustered behavior. While more complex editing behavior may be modeled in future studies based upon examining log files of configuration management system repositories, these three behavior patterns demonstrate the extremes and a middle behavior that clients may exhibit.

The client model maintains a state of either writing or reading and maintains a current position in the document. The client transitions between reading and writing according to the editing behavior being simulated (see above). Messages are sent to the network via an outbound message queue, and messages are received from the network via an inbound message queue.

The **network** is modeled to receive and send messages to and from the clients and the server. All messages within the system are modeled as strings with a source, destination, and payload so that each entity within the simulation knows that the message is designated for it. For the purposes of this simulation, we assume the time to transmit a message is consistent from each client and server to all other clients and servers, but we could easily create a lookup table within the network model to adjust costs dynamically based upon sender and recipient and bandwidth congestion. But such fidelity of the network was beyond the scope and interest of this research as we were interested in the number of messages, not the real-time performance of the network, especially since the network performance can vary considerably in different RTCES scenarios.

The network uses inbound and outbound message queues to receive and send messages from clients and servers.

The **server machine** is a complex model that consists of a *repository* model, a *server*, and a *lock proxy*. The *repository* is responsible for maintaining a set of documents/artifacts that can be checked in and out (similar

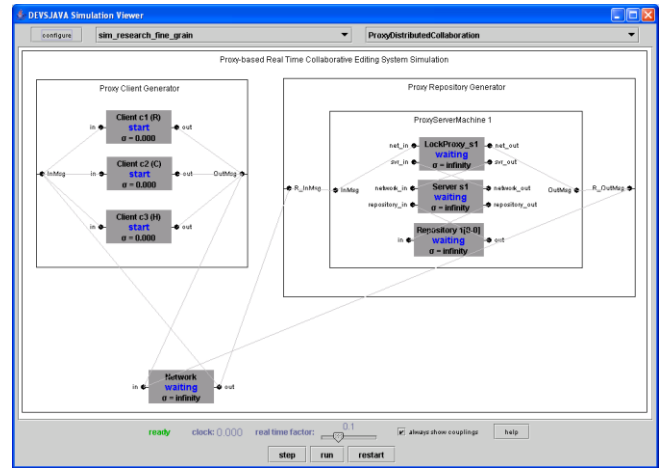


Figure 4: The Client, Network, and Server Models

to a standard configuration management system (CMS) like CVS or RCS). The *server* is responsible for receiving check-in and check-out requests and passing them to the repository; thus the server models a machine that would have a CMS running on it. The *lock proxy* is responsible for receiving messages from the network and parsing them to adjust the locks within the document tree. The lock proxy will only check out and check in a document if needed – thus it checks out and checks in document via proxy on behalf of the clients and keeps the server and repository ignorant that any complex management is taking place; as a result, we show how our dynamic lock management system can be added to existing repositories and easily increase their capabilities. Once checked out, the document is managed by the lock proxy and lock requests, lock releases, promotion/demotion, and OT-related messages are handled by the lock proxy and communicated to the clients.

The lock proxy model is the key model of the server machine model; this proxy model maintains the state of which documents are checked out of the server/repository models and maintains which users are present in each document and notifies clients upon promotion and demotion and passes on all OT-related messages to clients.

Additionally, we created complex models Proxy Client Generator and Proxy Repository Generator that allowed us to quickly create a set of client machines and a set of server machines; this was done to make it convenient to change the client behavior configuration and create multiple clients and multiple servers easily, but it does not affect the simulation as these complex models do not process messages or transition states.

For this study, we use a single server, but our models allow for distributing the repository of documents across multiple servers as we did in [13]. Figure 4 shows the models running within the DEVSJAVA Simulation Viewer; in this figure, there are three clients and one server machine.

3.3. Simulated Client and Using a Real Server

Having modeled both the clients and the server, we turn our attention to the implementation of the algorithms on the server. We implement the server so that it can be used in a real-world RTCES, but before employing it in a real-world scenario, we would like to validate that our simulation results in modeling the server accurately reflects the real performance that may be achieved when the server is fully implemented. In this scenario, we keep the client machine modeled as previously stated in Section 3.2. The simulated server machine is removed and we add a model called *OutConnection* that sends and receives messages to and from the real server using Web services invocations. The network is then connected to this new *OutConnection* model instead of the previous server machine model.

As previously stated, no other RTCES research has been able to so rigorously test their algorithms under a large-scale scenario with more than a handful of clients. Certainly others have measured performance of their algorithms with a large set of operations (see [8] for a recent example), but OT algorithmic studies focus on how quickly the algorithms may run and the storage capacities required; to date, no RTCES has been systematically tested with a large number of clients, as it is difficult to bring together so many users necessary for such a study. The impact of messages across the network has not been adequately measured in RTCES research, thus we address this cost by simulating a large number of clients connected to a real-world implementation of our server technology. As a result, we are able to determine how our system's performance scales as the number of clients increases.

3.4. Using a Real Client and Real Server

After the Web-services based server is implemented, we may begin development of a client application that connects to the server and allows multiple users to edit a shared document. The cursor position within the editor is tracked, and movement within the document automatically sends lock request and release messages to the server; as a result, clients are able to modify the shared document, and changes may be cached until a demotion message is received or the user leaves the space of the document that he owns. A preliminary version of this client editor application has been developed and is displayed in Figure 5. The dominant window (left) is the document's content, and the tree on the right shows the structure of the document tree based upon the document's content; the lower region shows state information such as in which section the cursor resides and displays messages from the server.

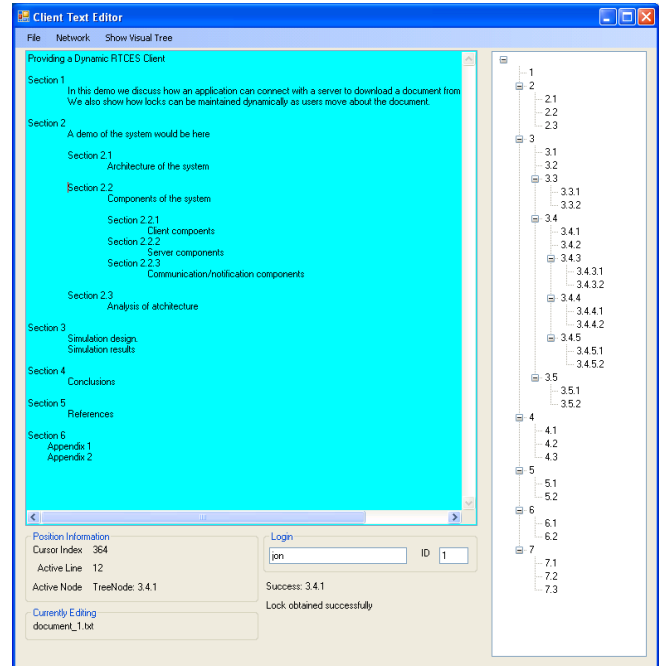


Figure 5 - The Client Editor

4. RESULTS AND ANALYSIS

We had previously measured message overhead and lock request success rates using our tree-based algorithms but without adopting OT within the tree [6]; the disadvantage of the previous approach is that locks were denied because another user already owned the leaf node requested at an average rate of 39%. By adopting OT into leaf nodes within the server simulated in this study, we remove all lock denials, thus all users are able to always edit the document. But we must address the additional communication cost associated with injecting OT into our system as clients must pass their changes to the other clients within the same section.

4.1. Event-based Simulation and Results

48 different runs of the simulation were recorded while modeling both the client and the server. There were six different document structures used in the simulations as shown in Table 1. Varying the structure of the document allows us to explore how varying the collaboration density (the ratio of users to leaves in the document structure) affects the messages generated in the simulation. Document structures 5 and 6 are representative of 4-page and 8-page conference papers respectively assuming the leaf nodes represent paragraphs.

Table 1 - Document Structure Types

| Document Structure | Number of Leaves | Maximum Depth | Average Depth |
|--------------------|------------------|---------------|---------------|
| 1 | 4 | 3 | 2.75 |
| 2 | 8 | 4 | 2.875 |
| 3 | 16 | 4 | 2.875 |
| 4 | 48 | 3 | 3 |
| 5 | 96 | 3 | 3 |
| 6 | 192 | 3 | 3 |

There were twelve configurations varying the number of clients and the document structure as shown in Table 2; for each of these twelve configurations, we ran simulations using four configurations of clients’ editing behavior configurations: all random, all clustered, all hybrid, and a uniform distribution among all three types. Thus there were 48 runs of the simulation total, and each simulation was run for 10,000 iterations.

While running the simulations, all message types (as defined in Section 2.1) were recorded as the clients made lock requests, updated their states, and notified other clients editing the same section of the document. As we had previously not utilized OT at the leaf nodes, we are particularly interested in how much communication overhead is due to adding OT to our system.

Table 2- Client/Document Configurations

| Simulation Configuration | Number of Clients | Document Structure |
|--------------------------|-------------------|--------------------|
| 1 | 3 | 1 |
| 2 | 9 | 1 |
| 3 | 3 | 2 |
| 4 | 9 | 2 |
| 5 | 3 | 3 |
| 6 | 9 | 3 |
| 7 | 3 | 4 |
| 8 | 9 | 4 |
| 9 | 3 | 5 |
| 10 | 9 | 5 |
| 11 | 3 | 6 |
| 12 | 9 | 6 |

We define the percentage of messages dealing with OT out of the total messages generated to be the *Dynamic OT Rate*. As shown in Figure 6, as the collaboration density (as measured by the ration of the number of clients and the number of leaves in the document) increases, the Dynamic OT Rate increases. Since collaboration density is directly proportional to how often users will share the same space within a document, it is natural to see the messages related to OT increase as collaboration density increases.

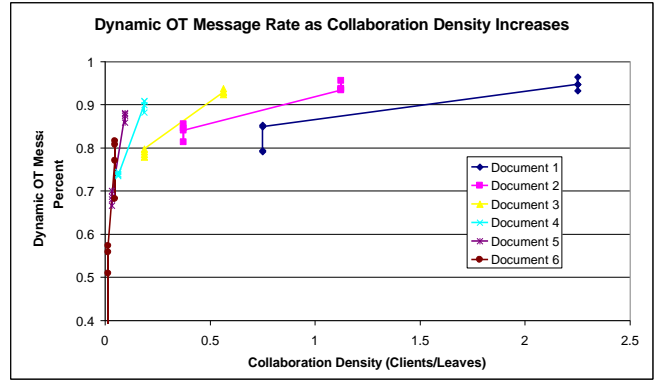


Figure 6 - Dynamic OT as Collaboration Increases

Figure 7 shows how our system employing dynamic locking and OT at the leaf level compares with using a “pure OT” (defined as broadcasting all changes to all users) performed with respect to communication for all 48 simulation configurations.

Since all messages are broadcast to all users other than the originating user in a pure OT system, we define the number of messages generated in a pure OT system as

$$M_{PureOT} = (n - 1)W$$

where n is the number of users and W is the number of write requests (the number of times users modified the document). Then the relative message overhead, M_o , of our dynamic lock OT system is defined as

$$M_o = \frac{LK + ULK + P + D + OTA + OTD + OTJ + OTM}{M_{PureOT}}$$

These message types were defined in Section 2.1. Note that we do not consider message types LK and ULK since they are the same in our dynamic system and a pure OT system.

Thus a relative message overhead of 1 reflects the dynamic lock with OT system incurs the same number of communication cost as a pure OT system. M_o above 1 reflects our system incurs more communication than a pure

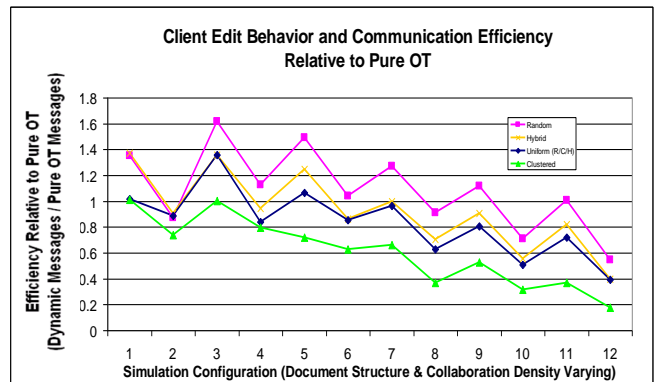


Figure 7 – Edit Behaviors and Communication Efficiency

OT system. M_o below 1 reflects our system incurs less communication than a pure OT system. Thus a lower value in Figure 7 is a reduction in communication costs.

From the results presented in Figure 7, it is clear that our system performs better relative to pure OT when all other variables remain the same and the number of clients increases (note that odd-even pairs reflect an increase from 3 to 9 in the number of clients). Additionally, when clients cluster their edit behavior, our system performs better relative to pure OT; this is intuitive in that the caching benefits of our system are better utilized when edits are localized/clustered. Further, the trend in Figure 7 shows that as the size of the document increases, our system increasingly outperforms pure OT.

5. CURRENT WORK

After the simulations involving modeled clients and a modeled server were performed, we adjust the simulation to interact with a real server. The client and network models remain exactly as previously described, but the added *OutConnection* model (see Section 3.3) connects with a real, fully-implemented server. In the real server, the algorithms for dynamic lock management are written in C# (to validate the open-systems approach we seek) and utilize a Web-service-based front-end so that any client can connect to and make use of the server's API.

Given that P, D, OTA and OTD messages are initiated in and must be sent from the server to the clients, the server maintains a set denoting which sections of the document each client resided (as done in our simulated version of the server). Consequently, the simulated *OutConnection* model subscribes to the server for future message notification, identifying itself with a socket (IP and port) so that the server can send messages to the simulation. This necessitates a "listener thread" running in the *OutConnection* model to receive any messages from the server; any server-initiated, incoming P, D, OTA and OTD messages are processed by the *OutConnection* model and sent to the client models.

We want to determine if our real server's performance would be comparable to the simulated server, so the same 48 runs of the simulation as described in Section 4.1 will be rerun using the client models connected to the real server. Additionally, we will measure round-trip times to complete each request to the server.

6. CONCLUSIONS AND FUTURE WORK

While previous work in RTCES has addressed ever-more effective OT algorithms for ensuring CCI, the costly communication overhead associated the global message broadcasts necessary on all changes within a pure OT system has not been heretofore researched. We have presented an improved system wherein our previous work in dynamic lock management has been further enhanced by adopting OT at a leaf level within the document tree. To verify our approach, we used DEVJSJAVA as a mechanism for implementing our design process – first modeling both the clients and the server and running the simulation, then implementing a real server and rerunning the simulation, then beginning work on a real client to connect to the real server.

The results we have achieved in this simulation research indicate that our system is more efficient with regard to communication costs than a pure OT system in any of the following scenarios: clustered editing behavior, large documents, and large numbers of users.

While OT was incorporated into the document tree in this study, we limited the use of OT to leaf nodes within the tree; it would be interesting to investigate how changing the depth at which OT is employed (i.e. allow OT to be adopted at a higher level in the tree) affects the communication efficiency. Our algorithms and data structures certainly allow for this, and we plan to investigate this in future work. It may prove that employing OT higher within the document tree reduces the aggregate computational cost since the lock promotion/demotion is reduced when OT is adopted.

Also, in this study, we do not explicitly measure the effects that changes to the document tree structure have on message overhead. Certainly users can add and remove sections of a document, and these changes would need to be transmitted to the server so that lock management could take place properly. Our previous work addressed how these changes to the structure of the tree can be handled [6], but the messaging overhead of such structure changes would not significantly affect the results of this study as we assume content modification dominates structure modification within the document; however, we may add this complexity into future simulations to create a more high-fidelity model of the client-server interactions.

Additionally, we are in the process of developing peer-to-peer versions of our algorithms such that there is no centralized server to create a bottleneck. Preliminary development shows much promise for this P2P approach, and we plan to further investigate how such an approach may further improve the communication and computation efficiencies within RTCES.

References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] Preston, J. A. and Prasad, S. K. "An Efficient Synchronous Collaborative Editing System Employing Dynamic Locking of Varying Granularity in Generalized Document Trees", *In Procs. 2nd Intl. Conf. on Collaborative Computing: Networking, Appln. and Worksharing*, Atlanta, Nov., 2006.
- [7] 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.
- [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] Oster, G., et al. "Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems", *In Procs. 2nd Intl. Conf. on Collaborative Computing: Networking, Appln. and Worksharing*, Atlanta, Nov., 2006.
- [10] Ignat, C. and Norrie, M, "Handling Conflicts through Multi-level Editing in Peer-to-peer Environments", *In 8th International Workshop on Collaborative Editing Systems*, Banff, Canada, November, 2006.
- [11] Davis, A. H., Sun, C., and Lu, J. "Generalizing Operational Transformation to the Standard General Markup Language", *Proceedings of CSCW 2002*, New Orleans, Louisiana, USA. November 16-20. pp. 58-67.
- [12] Hu, X. and Zeigler, B.P., "Model Continuity in the Design of Dynamic Distributed Real-Time Systems", *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans*, 35: 6, pp. 867- 878, November, 2005.
- [13] Preston, J. A. and Prasad, S. K. "A Web-Service-based Open-Systems Architecture for Achieving Heterogeneity in Synchronous Collaborative Editing Systems", *Proceedings of Cooperative Internet Computing 2006*, Hong Kong, October, 2006.