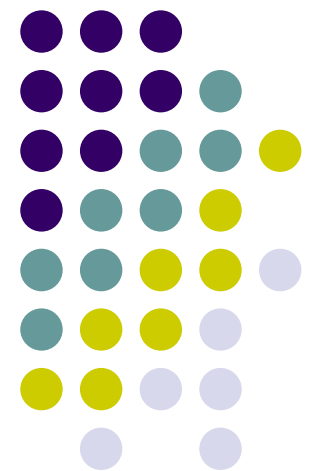


Dynamic Locking of Varying Granularity in Generalized Document Trees to Maximize Concurrency and Minimize Communication in Synchronous Collaborative Editing Systems

Jon A Preston
Sushil K Prasad
CollaborateCom 2006





Overview

- Motivation
- N-ary Document Tree
- Actions within the CES
- Algorithms
 - Obtaining Lock
 - Remove Lock
- Simulation and Communication Costs
- Tree Algorithms and OT
- Conclusions and Future Work



Motivation

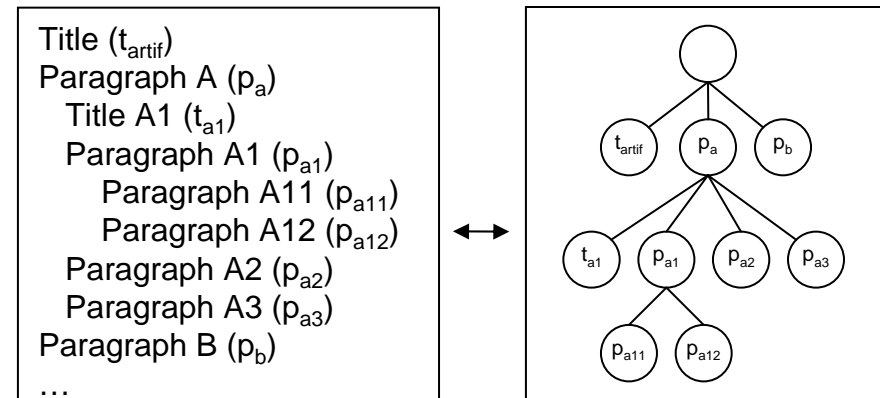
- Geographically distributed teams collaborating in real time
- Sharing a common document (or set of documents)
- Rapid changes/modification concurrently occurring

- Support synchronous collaborative editing
- Avoid problem of redundant/lost changes
- Avoid and/or enhance Operational Transformation (OT) and merging
- Minimize communication costs



N-ary Document Tree

- Model the document as an n-ary tree
- Sections and subsections
- Users can “own” a portion of the document without blocking other users from editing other portions of the document





Actions within the CES

<i>User Action</i>	<i>CES Tree Events</i>
Enter the CES	<ul style="list-style-type: none">• Place user as a reader in the default section of the document
Exit the CES	<ul style="list-style-type: none">• Remove the user from the CES and flush the user cache
Modify content within section A	<ul style="list-style-type: none">• OBTAINLOCK for section A• If not successful, deny the edit
Move from section A to section B	<ul style="list-style-type: none">• RELEASELOCK on section A• Place user as a reader in section B
Delete section A	<ul style="list-style-type: none">• OBTAINLOCK for section A• If successful, remove section A from tree
Combine section A and section B	<ul style="list-style-type: none">• OBTAINLOCK on section A• OBTAINLOCK on section B• If either fail, release any successfully obtained lock and deny the request• Else merge sections A and B in the tree (removing section B and RELEASELOCK on B)
Split section A into sections A and A'	<ul style="list-style-type: none">• OBTAINLOCK on section A• If not successful, deny the edit• Else create a new node A'' as a sibling of A, move content from A into A'



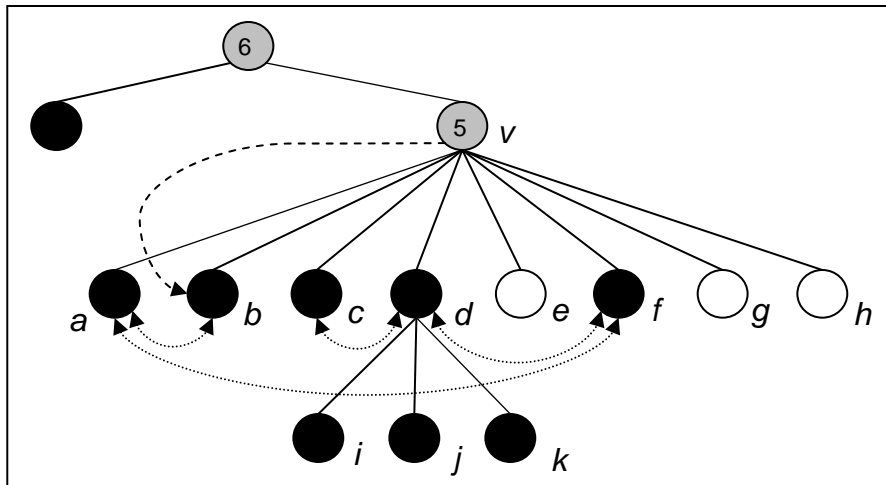
Algorithms

- ObtainLock
 - User wants to begin writing to (modify) a section of the document
- RemoveLock
 - User no longer writing and becomes a reader (moves to another section of the document or leaves the CES)
- Work top to bottom via handshake locks and are deadlock free
- Maintain node coloring to denote availability
 - Black – owned/locked
 - White – unowned/not locked
 - Grey – maintain “grey count” denoting number of conflicting users in subtrees below



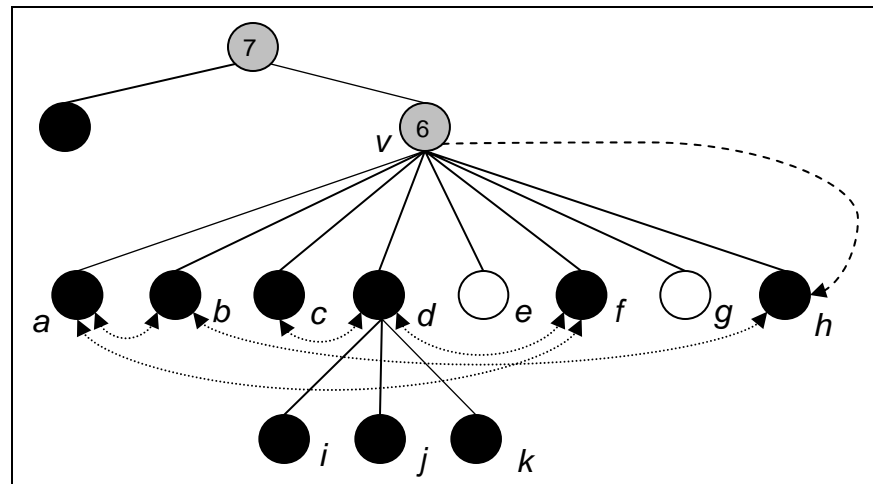
ObtainLock

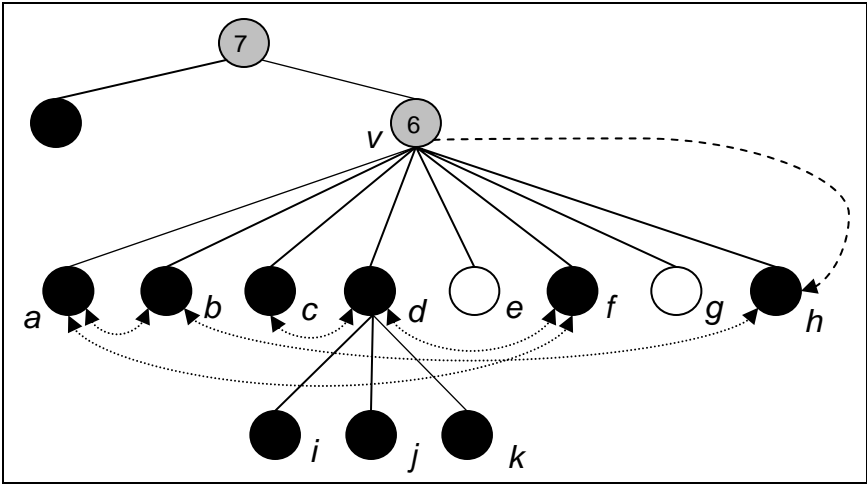
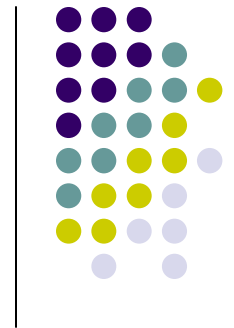
- Traverse top to bottom
- Increase “grey count” as you descend
- Keep descending until you reach a resolution node
 - Obtain an available node
 - Demote another user to resolve conflict
 - Deny request (or adopt OT)



ObtainLock(u_1, h)

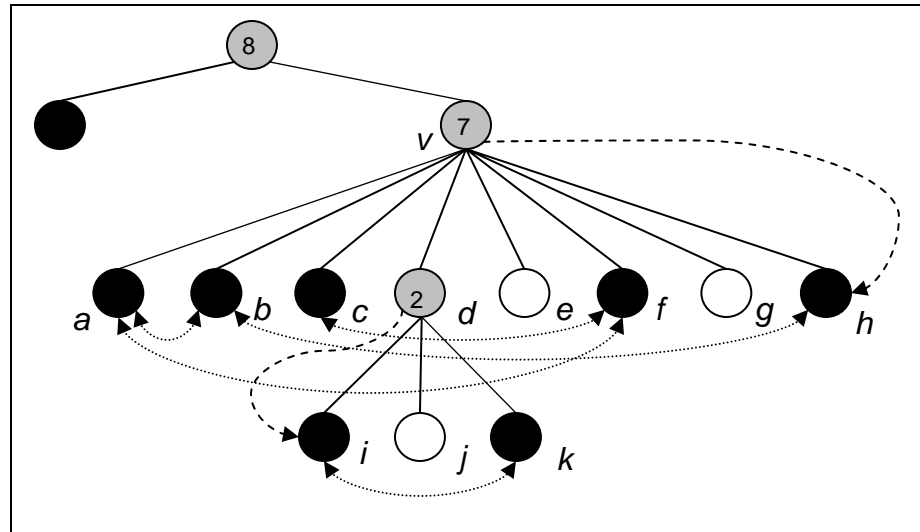
No Demotion





ObtainLock(u_2, k)

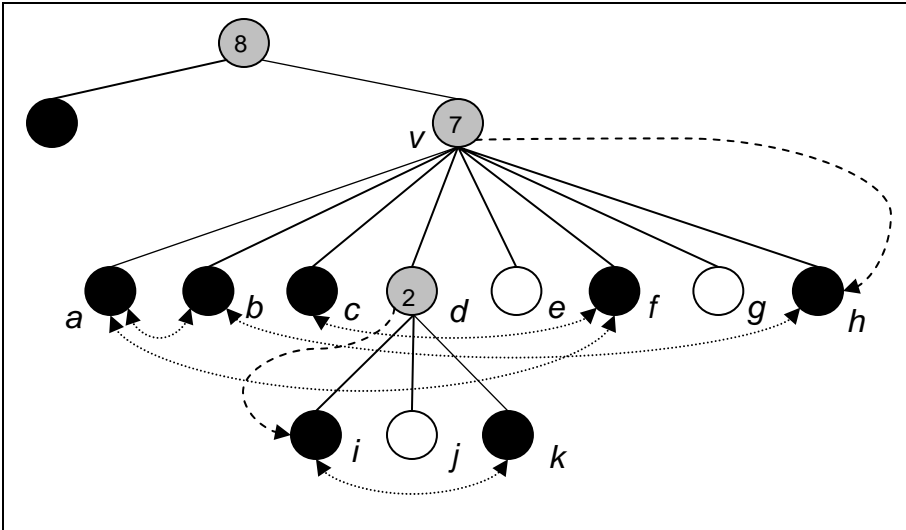
Demotion of u_1 from node d to node i





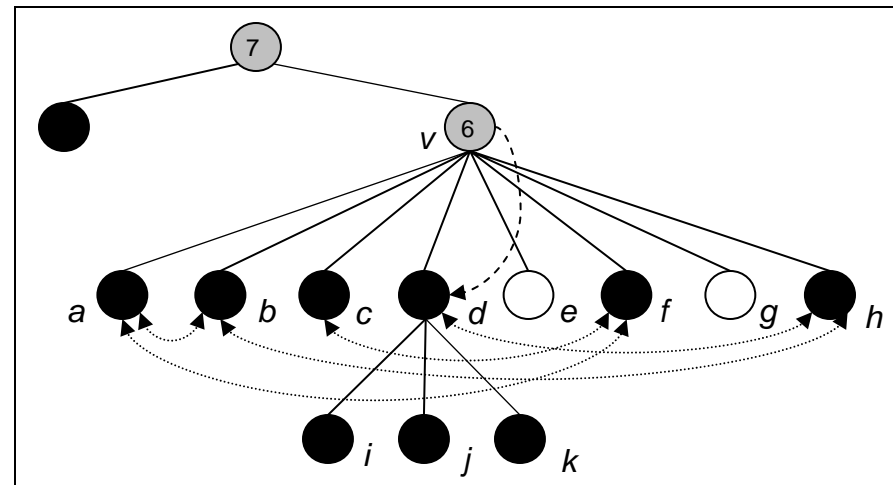
RemoveLock

- Traverse top to bottom
- Decrease “grey count” as you descend
- Keep descending until you reach a resolution node
 - Release owned node
 - Promote another user if conflict is now removed (“grey count” decreased to 1)



RemoveLock(u_1, i)

u_2 lock on node k is promoted to node d





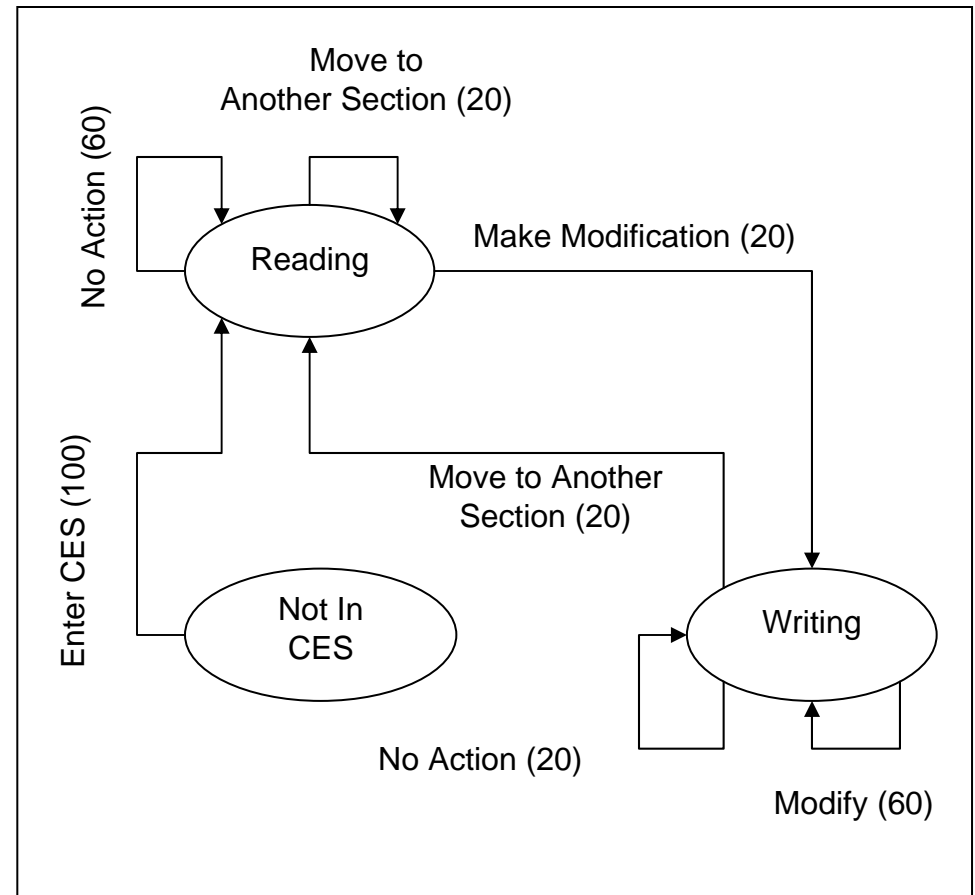
Simulation

- Discrete-event simulation
- Varied number of users/agents and document structure (number of sections)
- Logged events
 - Write events (how many edits were made to the document)
 - Communication among peers (writers to readers)
 - Communication among clients and server
 - Requests for document section
 - Requests to update cache
 - Promotion and Demotion (server to client communication)



Modeling Editing Behavior

- User/agent modeled as being in various states
- Probability of generating action at each time slice is dependant upon state (show in parenthesis)
- Simulation executed for 1000 time slices



Simulation Configurations and Communication Costs



<i>Configuration</i>			<i>Communication</i>						
# Agents	# Atomic Sections	Write Events	Dynamic Lock (DL) Messages				OT Messages	DL / OT Messages	DL Write Success Rate
			Client to Server	Server to Client (P/D)	Writer to Readers	TOTAL			
3	14	770	88	242	61	391	1540	25%	74.3%
6	14	1227	122	428	263	813	6135	13%	64.4%
9	14	1760	121	505	708	1334	14080	9.5%	61.6%
12	14	2004	144	615	1050	1809	22044	8.2%	56.7%
15	14	2542	154	731	1509	2394	35588	6.7%	55.8%
27	14	3434	92	856	4115	5063	89284	5.6%	46.0%
4	28	1004	108	326	53	487	3012	16%	73.3%
11	28	2349	253	775	425	1453	23490	6.2%	64.7%
18	28	3526	278	1040	1283	2601	59942	4.3%	62.2%
25	28	4530	289	1257	2430	3976	108720	3.7%	58.3%
32	28	5023	245	1381	3640	5266	155713	3.4%	52.8%

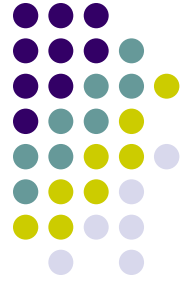
Client to Server: transitioning from writer to reader necessitates flushing cached modifications to server

Server to Client: P = Promotion; D = Demotion; lock update sent to client (adjust lock position/status)

Writer to Readers: incremental changes made by writer selectively multicast to readers within subsection

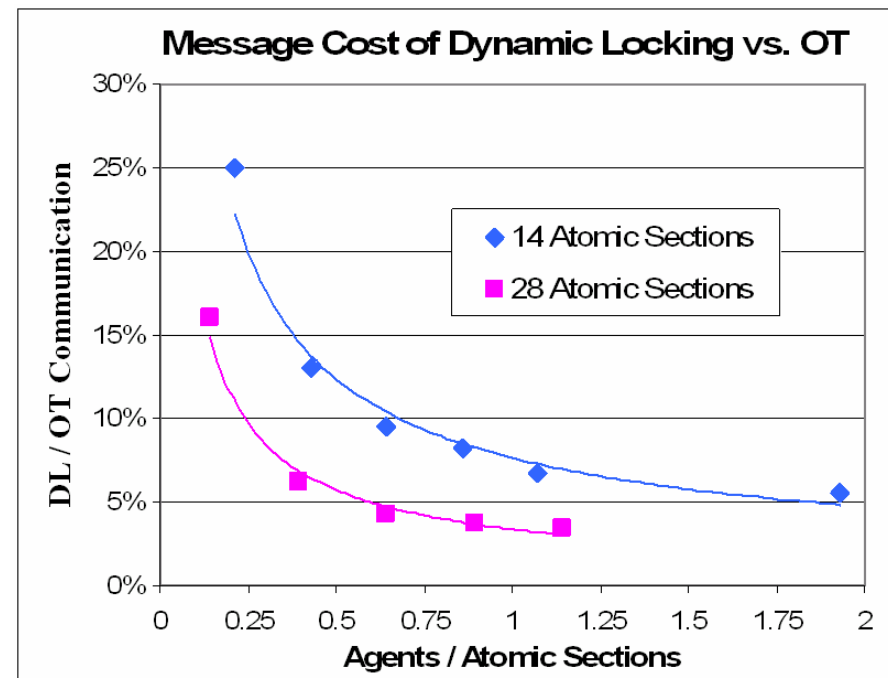
OT Messages: # of write events * (# agents – 1) (since we multicast to all agents other than the originating writer)

DL Write Success Rate: # successful modifications to document accomplished / total modifications attempted (only for the DL simulation since OT write success rate is by definition 100%)



Communication Costs

- OT multicast incurs high communication overhead relative to our cached, tree-based approach
- Tree-based communication costs ranged from 25% to less than 5% of the OT-based communication costs
- The improvement of tree-based over OT-based communication improves as the “density” of the collaboration increases





Tree Algorithms and OT

- Options to resolving conflict among multiple writers
 - Deny new writer request
 - Adopt OT at a sub-document level (atomic level)
- Cache changes locally when possible
 - No only one writer and no readers
- Selective multicast among all clients
 - Send updates to all readers
 - Send updates to writers and perform OT among writers
- Should decrease communication and computation cost versus traditional OT

Conclusions and Future Work



- Dynamic locking algorithms effective and efficient
- Change caching and selective multicast reduces communication costs over OT
- More formally/accurately model user editing patterns
- Sample/model real-world document structures
- Distribute the document tree among peers (P2P) to avoid single server bottleneck latency/starvation
- Further examine how OT may benefit from our dynamic document tree approach