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

<table>
<thead>
<tr>
<th>User Action</th>
<th>CES Tree Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter the CES</td>
<td>- Place user as a reader in the default section of the document</td>
</tr>
<tr>
<td>Exit the CES</td>
<td>- Remove the user from the CES and flush the user cache</td>
</tr>
<tr>
<td>Modify content within section A</td>
<td>- OBTAINLOCK for section A</td>
</tr>
<tr>
<td></td>
<td>- If not successful, deny the edit</td>
</tr>
<tr>
<td>Move from section A to section B</td>
<td>- RELEASELOCK on section A</td>
</tr>
<tr>
<td></td>
<td>- Place user as a reader in section B</td>
</tr>
<tr>
<td>Delete section A</td>
<td>- OBTAINLOCK for section A</td>
</tr>
<tr>
<td></td>
<td>- If successful, remove section A from tree</td>
</tr>
<tr>
<td>Combine section A and section B</td>
<td>- OBTAINLOCK on section A</td>
</tr>
<tr>
<td></td>
<td>- OBTAINLOCK on section B</td>
</tr>
<tr>
<td></td>
<td>- If either fail, release any successfully obtained lock and deny the request</td>
</tr>
<tr>
<td></td>
<td>- Else merge sections A and B in the tree (removing section B and RELEASELOCK on B)</td>
</tr>
<tr>
<td>Split section A into sections A and A’</td>
<td>- OBTAINLOCK on section A</td>
</tr>
<tr>
<td></td>
<td>- If not successful, deny the edit</td>
</tr>
<tr>
<td></td>
<td>- Else create a new node A’’ as a sibling of A, move content from A into A’’</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dynamic Lock (DL) Messages</td>
</tr>
<tr>
<td># Agents</td>
<td># Atomic Sections</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>27</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>18</td>
<td>28</td>
</tr>
<tr>
<td>25</td>
<td>28</td>
</tr>
<tr>
<td>32</td>
<td>28</td>
</tr>
</tbody>
</table>

- **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.

![Diagram showing message cost of dynamic locking vs. OT](image)
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