Concurrency Control and Awareness Support for Multi-synchronous Collaborative Editing

Mehdi Ahmed-Nacer, Valter Balegas,
Pascal Urso and Nuno Preguiça

University of Lorraine - LORIA Laboratory - France
Nova de Lisboa - Portugal

mehdi.ahmed-nacer@loria.fr

This work is partially funded by the french national research programs ConcoRDanT.

Collaborative editing applications

- Synchronous collaboration
  - Changes observed immediately
  - Merge concurrent updates by operations
- Asynchronous Collaboration
  - Changes observed after commit
  - Merge concurrent updates by states
- Synchronous application supports disconnected collaboration
  - Multi-synchronous applications
- Satisfy users intention during concurrent updates
- Keep awareness information to show to users the concurrent modifications

Limitations of existing Solutions
1- Google Drive

- Merge updates after disconnected periods
  - Semantic errors
  - Violatation of user intention
- Different problems can occur
  - Concurrent updates on the same sentence
  - Typographic errors
  - Cursor position
  - Update loss
- Limitations:
  - Mechanism provides less awareness information
    - Linearisation of the history
  - Keep *only* the previous revisions of the documents

Limitations of existing Solutions
2- Microsoft SkyDrive

- Users are forced to synchronize their documents explicitly
- Send the document to the server
- Solve conflicts manually if there were conflicting update
- Limitation:
  - Weak awareness and no synchronous update

Solution and idea

Goal

- Improve multi-synchronous collaborative editing application
  - Integrate concurrency control mechanisms
  - Keep more awareness information
  - Respect user intentions

How ?

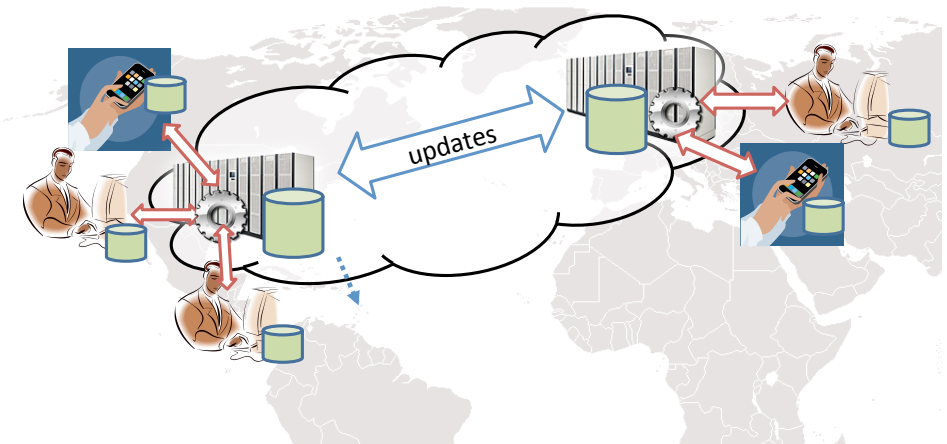
- Propose policies for handling conflicting operations
- Offer new operations
  - update and move
- Verify if the policies preserve the user intention

Conflict-free Replicated Data Types (CRDT)

- Class of distributed data type
- Modifications without coordination
- Replica converge to the same value when all updates are propagated
- Two types of CRDT:
  1. *Operation – based*: modifications are propagated as operations
  2. *State – based*: modifications are propagated as states
- *Operation – based* CRDT more adapted to synchronous collaboration

System Model

- Each node maintains a replica of the shared document
- Updates are propagated to all replica nodes
- Interface of document CRDT includes: insert, delete, update and move
- Updates are delivered in causal order
- Deployment in any architecture



Policy for Handling Conflicting Operations

	insert	update	delete	move
insert	keep two elements	not possible	not possible	not possible
	highlight new elems.	-	-	-
update	-	create versions	delete element	move the updated version
	-	show both versions	show del. element	highlight
delete	-	-	delete	delete element
	-	-	nothing needed	show del. element
move	-	-	-	create clones
	-	- -	- -	highlight clones

Table: Handling of concurrent updates to the same element and associated awareness solution

Structure

Positions	Documents
... <Pos2, id1> <Pos3, id3> <Pos4, id1> <Pos5, id2> ...	//variables int y; //variables int z;

elements	values
id1	{Pos2, Pos4}
	<(val_{11}, "//variables")>
id2	{Pos5}
	<(val_{21}, "int z;")>
id3	{Pos3}
	<(val_{31}, "int y;")>

Structure

Local: update(3, "int x;")
Remote: update(3, "int y=0;")

Positions	Documents
...	
<Pos2, id1>	//variables
<Pos3, id3>	~~int y;~~ int x;
<Pos4, id4>	//variables
<Pos5, id2>	int z;
...	

>>>>>>>local
int x;
==== origin
int y;
<<<<< remote
int y=0;

Awarness information

elements	values
id1	{Pos2, Pos4}
	(val1, "//variables")
id2	{Pos5}
	(val2, "int z;")
id3	{Pos3}
	<(val_{31}, "int x;"), (val_{32}, "int y=0;")>

Structure

operation: move(3, 6)

Positions	Documents
... <Pos2, id1> <Pos3, id3> <Pos4, id1> <Pos5, id2> ...	//variables int x; //variables int z;

elements	values
id1	{Pos2, Pos4}
	<(val11, "//variables")>
id2	{Pos5}
	<(val21, "int z;")>
id3	{Pos3}
	<(val31, "int x;")>

Structure

operation: move(3, 6)

Positions	Documents
...	
<Pos2, id1>	//variables
<Pos4, id1>	//variables
<Pos5, id2>	int z;
<Pos6, id3>	int x;
...	

elements	values
id1	{Pos2, Pos4}
	<(val11, "//variables")>
id2	{Pos5}
	<(val21, "int z;")>
id3	{Pos6}
	<(val31, "int x;")>

Structure

operation: del(2)

Positions	Documents
...	
~~<Pos2, id1>~~	~~//variables~~
<Pos4, id1>	//variables
<Pos5, id2>	int z;
<Pos6, id3>	int x;
...	

elements	values
id1	~~{Pos2~~, Pos4}
	<(val_{11}, "//variables")>
id2	{Pos5}
	<(val_{21}, "int z;")>
id3	{Pos6}
	<(val_{31}, "int x;")>

Evaluation Method

- Replays the history of Git repositories
  - By different op-based algorithms
- Transforms the state of the document to the set of operations
- Detects update and move operations
- Compare the number of modifications using our solution and another op-based algorithm

Operation Detection

- Compute distance of editions $\delta_{i,j}$ for each line
- Define Threshold Update (Tu) and Threshold Move(Tm)
  - $\delta_{i,j} <$Tu $\rightarrow$ update operation
  - $\delta_{i,j} <$Tm $\rightarrow$ move operation

```
- % test if x is greater than 0
- int a;
- Object toto;
- if (x > 0)
==================
+ % file procedure
+ % useful for stuff
+ % test if x is greater or equal than 0
+ int a=0;
+ File f;
+ if (x >= 0)
```

Tu= 0.3

Insert
```
+ % file procedure
+ % useful for stuff
```

Update
```
- % test if x is greater than 0
- int a;
==================
+ % test if x is greater or equal than 0
+ int a=0;
```

Delete
```
- Object toto;
```

Insert
```
+ File f;
```

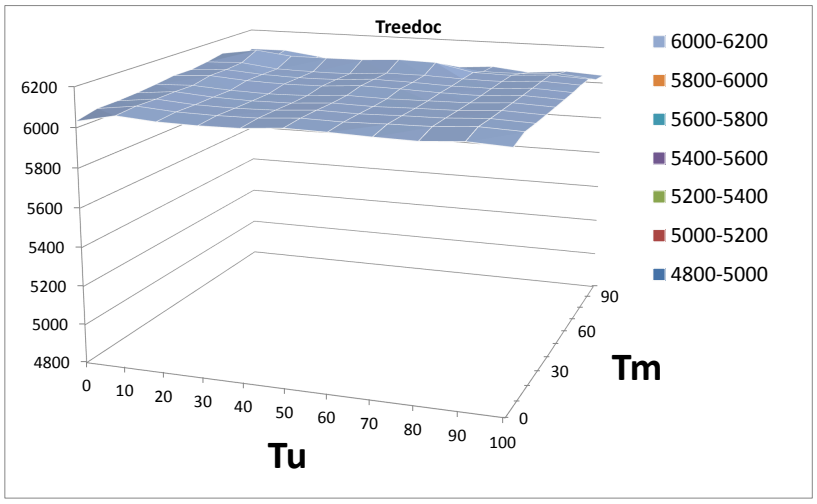
update
```
- if (x > 0)
==================
+ if (x >= 0)
```

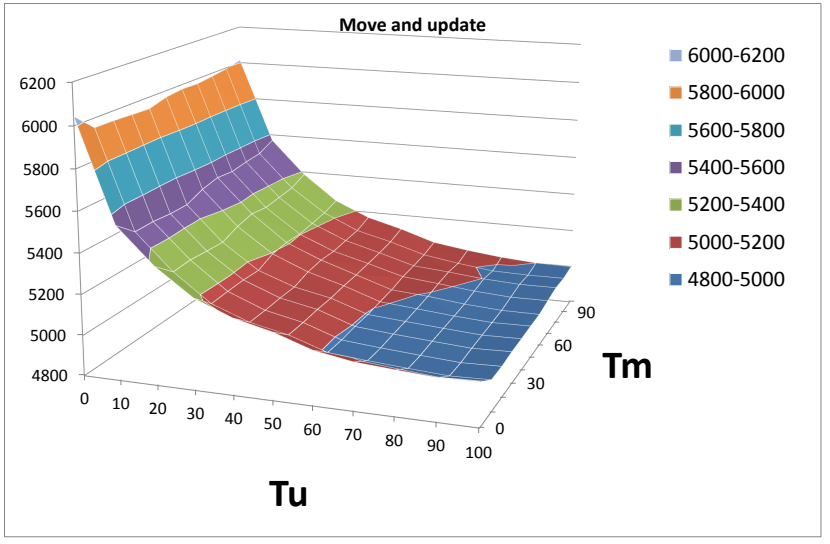
Experiment

- Estimate the adequate *Tu* and *Tm*
  1. Execute the algorithm in git/git repository
  2. Vary (Tu, Tm) from 0% to 100% in steps of 10%
- Compute the difference between user merges and automated merges computed by our algorithm
- Treedoc[1] is used as reference and compared with move/update algorithm

[1] Nuno P et al. A Commutative Replicated Data Type for Cooperative Editing

Results





- Best performance is obtained with Tu=0.9 and Tm=0.2
- Gain is 18% in git/git repository
- More results, see the paper !

Conclusion and perspectives

Conclusion

- Solution for supporting multi-synchronous collaborative editing
- Extend the traditional interface of document
  - Different granularity
  - Support move and update operations
- Keep more awareness information than traditional applications

Perspectives

- Introduce undo/redo mechanism
- Integrate our algorithm in a cloud-based web editing tool that supports geo-replication.

Thank you for your attention