Agenda

• Optimistic replication
  ▪ CVS, Subversion
  ▪ Duplicated databases
  ▪ Collaborative editing requirements
  ▪ Operational transformation: properties for convergence, transformation functions, algorithms
Collaborative editing: from users to community of users

“Isn’t it chaotic to all edit in the same document, even the same paragraph, at the same time?”

“Why would a group ever want to edit in the same line of text at the same time?”

GROVE, 1989
Collaborative editing: from users to community of users

2013: MOOC “Fundamentals of Online Education: Planning and Applications” with 40,000 participants
2016: Nuit debout, more than 70 people edit a pad
2018: online CSCW PC meeting with 120 members
Collaborative editing: from users to community of users

Real-time Wikipedia

Ponte Morandi: Revision history

View logs for this page (view filter log)

Search for revisions
From year (and earlier): 2018
From month (and earlier): all
Tag filter:

For any version listed below, click on its date to view it. For more help, see Help:Page history and Help:Edit summary.

External tools: Revision history statistics • Revision history search • Edits by user • Number of watchers • Page view statistics • Fix dead links

(cur) = difference from current version, (prev) = difference from preceding version, m = minor edit, e = section edit, a = automatic edit summary

(newest | oldest) View (newer 24 | older 24) (20 | 50 | 100 | 250 | 500)

Compare selected revisions

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)

(c) (t) (u) (e) (f) (r) (m) (l)
Limitations of Central Authority Systems

SCALABILITY

PRIVACY

Limitations of Central Authority Systems
Peer-to-Peer Collaborative Systems
Collaboration Modes

Concurrent Changes
Collaboration Modes

Offline Work
Collaboration Modes

Ad-hoc Collaboration
Operational transformation

- Domain of application: collaborative editing
- Document replication
  - Disconnected work
  - Better response time for real-time collaboration
Operational transformation

• Optimistic replication model
  ▪ An operation is:
    ○ Locally executed,
    ○ Sent to other sites,
    ○ Received by a site,
    ○ Transformed according to concurrent operations,
    ○ Executed on local copy

• 2 components:
  ▪ An integration algorithm: diffusion, integration
  ▪ Some transformation functions
Operational transformation

• Textual documents seen as a sequence of characters

• Operations
  ▪ ins(p,c)
  ▪ del(p)

• Three main issues
  ▪ Causality preservation
  ▪ Intention preservation
  ▪ Convergence
Causality

Site 1

Site 2

Site 3

$op_1 = ins(1, y)$

$op_2 = del(1)$

$op_2 = del(1)$

$op_1 = ins(1, y)$

$op_1 = ins(1, y)$

$op_2 = del(1)$

$op_1 = ins(1, y)$

$op_2 = del(1)$

$op_1 = ins(1, y)$

$op_2 = del(1)$

$op_1 = ins(1, y)$
Causality

\[ \begin{align*}
\text{Site 1} & \quad \text{Site 2} & \quad \text{Site 3} \\
[0,0,0] & \quad [0,0,0] & \quad [0,0,0] \\
\text{"yX"} & \quad \text{"yX"} & \quad \text{"y"} \\
\text{op}_1 = \text{ins}(1,y) & \quad \text{op}_1 = \text{ins}(1,y) & \quad \text{op}_1 = \text{ins}(1,y) \\
[1,0,0] & \quad [1,0,0] & \quad [1,0,0] \\
\text{"X"} & \quad \text{"X"} & \quad \text{"y"} \\
\text{op}_2 = \text{del}(1) & \quad \text{op}_2 = \text{del}(1) & \quad \text{op}_1 = \text{ins}(1,y) \\
[1,1,0] & \quad [1,1,0] & \quad [1,0,0] \\
\text{"X"} & \quad \text{"X"} & \quad \text{"y"} \\
\text{op}_2 = \text{del}(1) & \quad \text{delayed} \\
[1,1,0] & \\
\end{align*} \]
Intention

• Intention of an operation is the observed effect as result of its execution on its generation state

• Passing from initial state “ab” to final state “aXb” we can observe:
  - ins(2,X)
  - ins(a<X<b)
  - ins(a<X)
  - ins(X<b)
Preserving user intention (*)

• For any operation op, the effects of executing op at all sites should be the same as the intention of op.

• The effect of executing O does not change the effects of independent operations.

Intention violation

Site 1

"concurency contrl"

\[ \text{op}_1 = \text{ins}(7, r) \]

"concurency contrl"

Site 2

"concurency contrl"

\[ \text{op}_2 = \text{ins}(17, o) \]

"concurency control"

\[ \text{op}_2 = \text{ins}(17, o) \]

"concurency control"
Intention violation + divergence

Site 1

"concurrency control"

\( op_1 = \text{ins}(7, r) \)

Site 2

"concurrency control"

\( op_2 = \text{ins}(17, o) \)

"concurrency control"

\( op_2 = \text{ins}(17, o) \)

"concurrency control"

\( op_1 = \text{ins}(7, r) \)
Intention preservation

\[
T(\text{ins}(p_1, c_1), \text{ins}(p_2, c_2)) :\]
\[
\begin{align*}
\text{if } (p_1 < p_2) & \quad \text{return } \text{ins}(p_1, c_1) \\
\text{else} & \quad \text{return } \text{ins}(p_1 + 1, c_1)
\end{align*}
\]

\[\text{op}_1 = \text{ins}(7, r)\]
\[\text{op}_2 = \text{ins}(17, o)\]
\[\text{op}'_2 = \text{ins}(18, o)\]
Example transformation functions

\[ T(\text{ins}(p_1,c_1), \text{ins}(p_2,c_2)) :\]
\[
  \text{if } (p_1 < p_2) \quad \text{return } \text{ins}(p_1,c_1) \\
  \text{else return } \text{ins}(p_1+1,c_1)
\]

\[ T(\text{ins}(p_1,c_1), \text{del}(p_2)) :\]
\[
  \text{if } (p_1 \leq p_2) \quad \text{return } \text{ins}(p_1,c_1) \\
  \text{else return } \text{ins}(p_1-1,c_1) \\
  \text{endif}
\]

\[ T(\text{del}(p_1), \text{ins}(p_2,c_2)) :\]
\[
  \text{if } (p_1 < p_2) \quad \text{return } \text{del}(p_1) \\
  \text{else return } \text{del}(p_1+1)
\]

\[ T(\text{del}(p_1), \text{del}(p_2)) :\]
\[
  \text{if } (p_1 < p_2) \quad \text{return } \text{del}(p_1) \\
  \text{else if } (p_1 > p_2) \quad \text{return } \text{del}(p_1-1) \\
  \text{else return } id()
\]
Convergence but no intention preservation

Thomas Write Rule

Site 1

\( \text{op}_1 = \text{set}(s, \text{"AXB"}) \)

\( \{s, \text{"AXB"}, f, (0h, 1), (9h01, 1)\} \)

\( \{s, \text{"AB"}, f, (0h, 1), (0h, 1)\} \)

Site 2

\( \{s, \text{"AB"}, f, (0h, 1), (0h, 1)\} \)

Site 3

\( \{s, \text{"AB"}, f, (0h, 1), (0h, 1)\} \)

\( \text{op}_2 = \text{set}(s, \text{"AYB"}) \)

\( \{s, \text{"AYB"}, f, (0h, 1), (9h02, 3)\} \)

\( \{s, \text{"AYB"}, f, (0h, 1), (9h02, 3)\} \)

\( \{s, \text{"AYB"}, f, (0h, 1), (9h02, 3)\} \)
Convergence – TP1 property

- \( T(\text{op}_2: \text{operation}, \text{op}_1: \text{operation}) = \text{op}'_2 \)
  - \( \text{op}_1 \) and \( \text{op}_2 \) concurrent, defined on a state \( S \)
  - \( \text{op}'_2 \) same effects as \( \text{op}_2 \), defined on \( S.\text{op}_1 \)

\[
[\text{TP1}] \quad \text{op}_1 \circ T(\text{op}_2, \text{op}_1) \equiv \text{op}_2 \circ T(\text{op}_1, \text{op}_2)
\]
Convergence – TP2 property

\[ TP2 \quad T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2)) \]
OT Problems

- Design and verify Transformation functions $T$
- $T$ also known as transpose_fd
- Verification of conditions TP1 and TP2
  - Combinatorial explosion ($>100$ cases for a string)
  - Iterative process
  - Repetitive and error prone task
Partial concurrency

Site 1

op₁ = ins(5, p)

op₂ = del(5)

Site 2

op’₁ = ins(5, p)

op’’₂ = del(7)

T(op₂, op₁) = del(6)

T(op’ ₂, op₃) = del(7)

T(op₁, op₂) = ins(5)

T(op₃, op₂) not allowed to be performed !!!
Partial concurrency

Site 1

“telefone”

$op_1 = \text{ins}(5, p)$

“telefone”

$op_3 = \text{ins}(6, h)$

“telefone”

$op''_2 = \text{del}(7)$

“telephone”

Site 2

“telefone”

$op_2 = \text{del}(5)$

“telefone”

“telefone”

 transpose_bk

Site 2

“telefone”

$op_1 = \text{ins}(5, p)$

“telefone”

$op'_1 = \text{ins}(5, p)$

“telefone”

$op'_2 = \text{T}(op_2, op_1) = \text{del}(6)$

“telefone”

$op'_3 = \text{T}(op_3, op'_2) = \text{ins}(6, h)$

“telefone”

“telephone”

“telefone”

“telefone”

“telefone”

“telefone”

“telefone”
Partial concurrency

- transpose\_bk(op_1, op'\_2) = (op_2, op'\_1)
  - op'\_2 = T(op_2, op_1)
    Therefore op_2 = T^{-1}(op'\_2, op_1)
  - op'\_1 = T(op_1, op_2)
Example transformation functions

\[ T(\text{ins}(p_1, c_1), \text{ins}(p_2, c_2)) :- \]
\[ \text{if } (p_1 < p_2) \text{ return } \text{ins}(p_1, c_1) \]
\[ \text{else return } \text{ins}(p_1+1, c_1) \]

\[ T(\text{ins}(p_1, c_1), \text{del}(p_2)) :- \]
\[ \text{if } (p_1 \leq p_2) \text{ return } \text{ins}(p_1, c_1) \]
\[ \text{else return } \text{ins}(p_1-1, c_1) \]
\[ \text{endif} \]

\[ T(\text{del}(p_1), \text{ins}(p_2, c_2)) :- \]
\[ \text{if } (p_1 < p_2) \text{ return } \text{del}(p_1) \]
\[ \text{else return } \text{del}(p_1+1) \]

\[ T(\text{del}(p_1), \text{del}(p_2)) :- \]
\[ \text{if } (p_1 < p_2) \text{ return } \text{del}(p_1) \]
\[ \text{else if } (p_1 > p_2) \text{ return } \text{del}(p_1-1) \]
\[ \text{else return } \text{id()} \]

TP1 not respected!
Ressel transformation functions (*)

\[ T(\text{ins}(p_1,c_1,u_1), \text{ins}(p_2,c_2,u_2)) :- \]
  \[ \text{if } ((p_1<p_2) \text{ or } (p_1=p_2 \text{ and } u_1<u_2)) \text{ return } \text{ins}(p_1,c_1,u_1) \]
  \[ \text{else return } \text{ins}(p_1+1,c_1,u_1) \]

\[ T(\text{ins}(p_1,c_1,u_1), \text{del}(p_2,u_2)) :- \]
  \[ \text{if } (p_1\leq p_2) \text{ return } \text{ins}(p_1,c_1,u_1) \]
  \[ \text{else return } \text{ins}(p_1-1,c_1,u_1) \]
  \[ \text{endif} \]

\[ T(\text{del}(p_1,u_1), \text{ins}(p_2,c_2,u_2)) :- \]
  \[ \text{if } (p_1<p_2) \text{ return } \text{del}(p_1,u_1) \]
  \[ \text{else return } \text{del}(p_1+1,u_1) \]

\[ T(\text{del}(p_1,u_1), \text{del}(p_2,u_2)) :- \]
  \[ \text{if } (p_1<p_2) \text{ return } \text{del}(p_1,u_1) \]
  \[ \text{else if } (p_1>p_2) \text{ return } \text{del}(p_1-1,u_1) \]
  \[ \text{else return } \text{id}() \]

$TP1$ ok, but not $TP2$ !
False-tie problem

Site 1
"abc"

Site 2
"abc"

Site 3
"abc"

$\text{op}_1 = \text{Insert}(2, x)$

$\text{op}_2 = \text{Delete}(2, b)$

$\text{op}_3 = \text{Insert}(3, y)$

"axbc"

"ac"

"abyc"

$\text{op'}_1 = \text{Insert}(2, x)$

$\text{op'}_3 = \text{Insert}(2, y)$

"axyc"?

"ayxc"?
**TTF (Tombstone Transformation Functions) Approach (*)**

- Keep “tombstones” of deleted elements

---

Tombstone Transformation Functions

- $T(\text{insert}(p_1, e_1, s_1), \text{insert}(p_2, e_2, s_2))$
  
  ```
  if(p_1 < p_2) return \text{insert}(p_1, e_1, s_1)
  else if(p_1 = p_2 \text{ and } s_1 < s_2) return \text{insert}(p_1, e_1, s_1)
  else return \text{insert}(p_1 + 1, e_1, s_1)
  ```

- $T(\text{insert}(p_1, e_1, s_1), \text{delete}(p_2, e_2, s_2))$
  
  ```
  return \text{insert}(p_1, e_1, s_1)
  ```

- $T(\text{delete}(p_1, s_1), \text{insert}(p_2, s_2))$
  
  ```
  if(p_1 < p_2) return \text{delete}(p_1, s_1)
  else return \text{delete}(p_1 + 1, s_1)
  ```

- $T(\text{delete}(p_1, s_1), \text{delete}(p_2, s_2))$
  
  ```
  return \text{delete}(p_1, s_1)
  ```
Compacted storage model

- Compacted model = sequence of (character, abs_pos)
Delta storage model

- Delta model = sequence of (character, offset)
Models comparison

• Basic Model
  ▪ Deleted characters are kept
  ▪ Size of the model is growing infinitely

• Compacted Model
  ▪ Update absolute position of all characters located after the effect position

• Delta Model
  ▪ Update the offset of next character

• Our observations
  ▪ View2model can be optimised (caret position)
  ▪ Overhead of view2model is not significant
SOCT2 algorithm (*)
General control algorithm

a) The initial history buffer

Equivalent HB

b) Principle of integration

GOT algorithm(*)

- Does not need to satisfy TP1 and TP2
- Requires a global serialisation order
  - Sum of state vector components
  - If equality, then priority on sites
- Requires undo/redo mechanism
- Undo/redo very costly

SOCT4 algorithm(*)

- Does not use undo/redo mechanism
- Eliminates TP2, but requires TP1
- Does not need state vectors
- Global order of operations according to timestamps generated by a sequencer
- Local operations executed immediately
- Assigns a timestamp to the operation and transmits it to the other sites
- Defers broadcast until all preceding operations were executed
- Transformations performed by each site

SOCT4 algorithm

Histoire $H_S(n)$ avant l’intégration $op_1$ $op_2$ ... $op_i$

transposition en avant de $op_{i+1}$

Intégration de $op_{i+1}$ $op_1$ $op_2$ ... $op_i$

transpositions en avant des opérations locales

History $H_S(n)$ après intégration $op_1$ $op_2$ ... $op_i$ $op_{i+1}$ $op'_{L_1}$ $op'_{L_m}$

opérations locales transposées en avant

opérations locales en attente de diffusion

$op_{i+1}$ $op_{L_1}$ $op_{L_m}$

seq
So6- variation of SOCT4 algorithm(*)

Sync \( \langle \log, N_s \rangle \) :-
  while \( \left( \left( op_r = \text{getOp}(N_s+1) \right) \neq \emptyset \right) \)
  for ( \( i = 0; i < \log\text{.size}(); i + + \) )
    \( op_l = \log \langle i \rangle \); 
    \( \log \langle i \rangle = T(op_l, op_r) \)
    \( op_r = T(op_r, op_l) \); 
  endfor
  execute \( op_r \)
  \( N_s = N_s + 1 \)
endwhile

for ( \( i = 0; i < \log\text{.size}(); i + + \) )
  \( op'_l = \log \langle i \rangle \); 
  if \( \text{send} \langle op'_l, N_s + 1 \rangle \) then 
    \( N_s = N_s + 1 \)
  else 
    error \ 'need to synchronize' \ 
  endif
endfor

\( \text{getOp}(ticket) \) retrieves operation identified by timestamp \( ticket \)
\( \text{send}(op, ticket) \) sends local operation with timestamp \( ticket \). If ticket already exists (a concurrent synchronization is in progress), returns false

So6 algorithm

<table>
<thead>
<tr>
<th>Site1, Ns=0</th>
<th>Site2, Ns=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1</td>
<td>op3</td>
</tr>
<tr>
<td>op2</td>
<td>op4</td>
</tr>
<tr>
<td>s1=synchronize</td>
<td>s2=synchronize</td>
</tr>
</tbody>
</table>

- At s1
  - sync([op1,op2],0)
  - Send op1, op2
  - Ns=2

- At s2
  - sync([op3,op4],0)
  - op’1=T(op1,op3)
  - op’3=T(op3,op1)
  - op’’1=T(op’1,op4)
  - op’4=T(op4,op’1)
  - op’2=T(op2,op’3)
  - op’’3=T(op’3,op2)
  - op’’2=T(op’2,op’4)
  - op’’4=T(op’4,op’2)
  - op’’1, op’’2 are executed
  - op’’3, op’’4 are sent
  - Ns=4
## SO6 algorithm

<table>
<thead>
<tr>
<th>Site 1, $Ns=0$</th>
<th>Site 2, $Ns=0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1</td>
<td>op3</td>
</tr>
<tr>
<td>op2</td>
<td>op4</td>
</tr>
<tr>
<td>$s1$=synchronize</td>
<td></td>
</tr>
<tr>
<td>$s2$=synchronize</td>
<td></td>
</tr>
<tr>
<td>$s3$=synchronize</td>
<td></td>
</tr>
</tbody>
</table>

### Site 1

| op1 | op2 | $op''3=T(T(op3,op1),op2)$ | $op''4=T(T(op4,op'1),op'2)$ |

### Site 2

| op3 | op4 | $op''1=T(T(op1,op3),op4)$ | $op''2=T(T(op2,op'3),op'4)$ |
SO6 algorithm

- create(obj)
- change_color(obj,color)
- del(obj)

```
• T(create(obj), _):-
  return create(obj)

• T(change_color(obj1, c), del(obj2)): -
  if (obj1 == obj2) then return id()
  else return change_color(obj1, c)

• T(del(obj1), del(obj2)): -
  if (obj1 == obj2) then return id()
  else return del(obj1)

• T(del(obj1), change_color(obj2, c)): -
  return del(obj1)

• T(change_color(obj1, c1), change(obj2, c2)): -
  if (obj1 == obj2) then
    if (c1 > c2) then return change_color(obj1, c1)
    else return id()
  else return change_color(obj1, c1)

• T(id(), _): - return id()
• T(op, id()): - return op
```
\[ \text{Site 1} \]

- \( \text{op1} = \text{del}(\text{obj1}) \)
- \( \text{op2} = \text{change\_color}(\text{obj2}, \text{red}) \)
- \( \text{op'''3} = \text{id}() \)
- \( \text{op'''4} = \text{id}() \)

\[ \text{Site 2} \]

- \( \text{op3} = \text{change\_color}(\text{obj1}, \text{green}) \)
- \( \text{op4} = \text{change\_color}(\text{obj2}, \text{blue}) \)
- \( \text{op'''1} = \text{del}(\text{obj1}) \)
- \( \text{op'''2} = \text{change\_color}(\text{obj2}, \text{red}) \)

\[ \text{red} > \text{blue} \]

- \( \text{op1} = T(\text{op1}, \text{op3}) = \text{del}(\text{obj1}) \)
- \( \text{op'''3} = T(\text{op3}, \text{op1}) = \text{id}() \)
- \( \text{op'''1} = T(\text{op'''1}, \text{op4}) = \text{del}(\text{obj1}) \)
- \( \text{op'''4} = T(\text{op4}, \text{op'''1}) = \text{change\_color}(\text{obj2}, \text{blue}) \)
- \( \text{op2} = T(\text{op2}, \text{op'''3}) = \text{change\_color}(\text{obj2}, \text{red}) \)
- \( \text{op'''3} = T(\text{op'''3}, \text{op2}) = \text{id}() \)
- \( \text{op'''2} = T(\text{op'''2}, \text{op'''4}) = \text{change\_color}(\text{obj2}, \text{red}) \)
- \( \text{op'''4} = T(\text{op'''4}, \text{op2}) = \text{id}() \)
Jupiter algorithm(*)

- Used in Google Drive
- Requires a central server
- Eliminates TP2, but requires TP1
- Does not need state vectors
- Transformations done on the server + client side

Jupiter algorithm

- \( \text{xform}(c,s) = \{c', s'\} \)
- \( \text{xform}(\text{del } x, \text{del } y) = \)
  \[
  \begin{cases} 
  \{\text{del } x-1, \text{del } y\} & \text{if } x>y \\
  \{\text{del } x, \text{del } y-1\} & \text{if } x<y \\
  \{\text{no-op, no-op}\} & \text{if } x=y 
  \end{cases}
  \]
Jupiter algorithm
Jupiter algorithm

client

server

\( x, y \)

\( x+k, y \)

\( x+i, y \)

\( x+i, y+1 \)
Jupiter algorithm
2 sites

```c
int myMsgs = 0; /* number of messages generated */
int otherMsgs = 0; /* number of messages received */
queue outgoing = {};

Generate(op) {
    apply op locally;
    send(op, myMsgs, otherMsgs);
    add (op, myMsgs) to outgoing;
    myMsgs = myMsgs + 1;
}

Receive(msg) {
    /* Discard acknowledged messages. */
    for m in (outgoing) {
        if (m.myMsgs < msg.otherMsgs)
            remove m from outgoing
    }
    /* ASSERT msg.myMsgs == otherMsgs. */
    for i in [1..length(outgoing)] {
        /* Transform new message and the ones in the queue. */
        {msg, outgoing[i]} = xform(msg, outgoing[i]);
    }
    apply msg.op locally;
    otherMsgs = otherMsgs + 1;
}
```
\[ \text{op}^3 = T(T(\text{op}3, \text{op}1), \text{op}2) \]

outgoing = \[T(\text{op}3, \text{op}1), T(\text{op}2, T(\text{op}3, \text{op}1))\]

(2, 1)

\[ \text{Exec. op'} 3 \]

\[ \text{op'} 4 = T(\text{op}4, T(\text{op}2, T(\text{op}3, \text{op}1)) \]

outgoing = \[T(T(\text{op}2, T(\text{op}3, \text{op}1)), \text{op}4)\]

(2, 2)

\[ \text{Exec. op'} 4 \]
Jupiter algorithm – generalisation n

Clients

Client 1

Client 2

Client 3

apply msg.op locally;

Algorithm changes at server side

apply msg.op locally;
for (c in client list) {
    if (c != client)
        send(c, msg);
}
Jupiter algorithm

- Requires a server that performs transformations
- Not suitable for P2P environments
- False tie scenario gives different results according to integration order

Site 1  “abc”

op₁=Insert(2,x)

“axbc”

Site 2  “abc”

op₂=Delete (2,b)

“ac”

Site 3  “abc”

op₃=Insert(3,y)

“abyc”

op’₁=Insert(2,x)

“axyc”?

op’₃=Insert(2,y)

“ayxc”? 