OPERATION TRANSFORMATION

Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill
dewan@cs.unc.edu
IMMEDIATELY DELIVERING CONCURRENT MESSAGES

- A tree of message paths exists
- Create vector time stamp and buffer for each leaf in the path
- When a message arrives see if its vector time stamp > one of the vector time stamps, put in the buffer for that vector time stamp
- Otherwise create a new vector time stamp and buffer (VectorTimeStampCopiedAndNewBufferCreated) deliver the message after flagging concurrency
**Concurrency: Different Successor**

$v^1 = (a^1, .. a^n)$ is a successor of

There exists $1 \leq i \leq n$, $a^i = 1 + b^i$

for all $j \neq i$, $a^j = b^j$

Not successor!
**Concurrent Successors**

- $v^1 = (a^1, .. a^n)$ is a successor of $v^2 = (b^1, .. b^n)$

There exists $1 \leq i \leq n$, $a^i = 1 + b^i$ for all $j \neq i$, $a^j \leq b^j$

Need to buffer messages along each path
DELIVERING CONCURRENT MESSAGES

- A DAG of message paths exist
- Create buffer for each leaf in the DAG
- When a message arrives find a buffer
  - See if a buffer exists in which the head message is not concurrent with this message
  - Otherwise create a new buffer
- Insert message at appropriate position in buffer
- Process buffer as before but now use a different successor function
  - \((b^1, b^2, \ldots, b^n)\) is successor of \((a^1, a^2, \ldots, a^n)\) if for some \(j\), \(b^j = a^j + 1\) and for all \(i \neq j\), \(b^i \leq a^i\)
  - Takes into account that a message on a different path from an ancestor of the current time stamp node arrived
DELIVERING CONCURRENT MESSAGES – FULL SEARCH

- A DAG of message paths exist
- Create a single buffer for all messages
- When a message arrives put it in the buffer at some position
- Process buffer as before
  - but now use a different successor function
    - \((b^1, b^2, \ldots, b^n)\) is successor of \((a^1, a^2, \ldots, a^n)\) if for some \(j\), \(b^j = a^j + 1\) and for all \(i \neq j\), \(b^i \leq a^i\)
    - Takes into account that a message on a different path from an ancestor of the current time stamp node arrived
  - Search the entire buffer rather than look at the head of the buffer
OPERATION TRANSFORMATION
Delivering Concurrent Messages – Full Search

- A DAG of message paths exist
- Create a single buffer for all messages
- When a message arrives put it in the buffer at some position
- Process buffer as before
  - but now use a different successor function
    - \((b^1, b^2, \ldots, b^n)\) is successor of \((a^1, a^2, \ldots, a^n)\) if for some \(j\), \(b^i = a^j + 1\) and for all \(i \neq j\), \(b^i \leq a^i\)
    - Takes into account that a message on a different path from an ancestor of the current time stamp node arrived
  - Search the entire buffer rather than look at the head of the buffer

Are causality guarantees among concurrent paths enough?
CONCURRENT EDITING: INITIAL STATE
CONCURRENT INSERTIONS
IMMEDIATELY DELIVERING CONFLICTING REMOTE OPERATION

PC 1

I, 2, u

I, 5, ?

PC 2

I, 2, u

1 2 3 4 5 6

1 2 3 4 5 6

? h

? h
**Partially Ordered Vector Time Stamps**

\( v = (x^1, \ldots, x^n) \) at Site \( S^i \) \( \rightarrow \) Site \( S^j \) has received \( x^i \) messages from Site \( S^i \) for all \( 1 \leq i \leq n \)

<table>
<thead>
<tr>
<th>( v^1 )</th>
<th>( v^2 )</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v^1 = (a^1, \ldots, a^n) )</td>
<td>( v^2 = (b^1, \ldots, b^n) )</td>
<td>for all ( 1 \leq i \leq n ), ( a^i == b^i )</td>
</tr>
<tr>
<td>( v^1 = (a^1, \ldots, a^n) )</td>
<td>( v^2 = (b^1, \ldots, b^n) )</td>
<td>for all ( 1 \leq i \leq n ), ( a^i \leq b^i )</td>
</tr>
<tr>
<td>( v^1 = (a^1, \ldots, a^n) )</td>
<td>( v^2 = (b^1, \ldots, b^n) )</td>
<td>for some ( 1 \leq i \leq n ), ( a^i &lt; b^i )</td>
</tr>
<tr>
<td>( v^1 = (a^1, \ldots, a^n) )</td>
<td>( v^2 = (b^1, \ldots, b^n) )</td>
<td>for some ( 1 \leq j \leq n ), ( a^i &gt; b^i )</td>
</tr>
</tbody>
</table>

Concurrent Input
Causal time stamps allow computer to determine concurrent actions.

Do not impose a common or total order, which inherently does not exist.

Total order often important.

Broadcast supporting total order called atomic broadcast.
**Asynchronous Broadcast**

1. Perform operation $o$

2. toOthers send the operation using peer to peer communication

3. Perform received operation (in causal order)

---

**PC 1**

- Done?

**PC 2**

- Done?

**PC 3**

- Done?

---

No coordination before performing operation

Atomic asynchronous, application-unaware broadcast impossible

Synchronous and Coordination?
1. Perform operation on PC 1
2. toAll send the operation using relayed communication
3. perform operation on its receipt

Each site performs operations in the same sequence assuming ordered unicast

Delay (extra hop) state of operation issue not same as state of execution, though all sites are consistent
ORDERING WITH ATOMIC BROADCAST
FIRST OPERATION EXECUTES
SECOND OPERATION EXECUTES

Common state but “intention” violation

Context of operation not the same as when it was issued

Worse outcome than intention violation?
CONCURRENT INTERACTION: DELETE, MODIFY
ATOMIC BROADCAST ORDERING
FIRST OPERATION EXECUTES

PC 1

M,1, L

PC 2

M,1, L
SECOND OPERATION CAUSES EXCEPTION

Common behavior but exception

Context of operation not the same as when it was issued

Can concurrency control explain or fix this?

ArrayIndexOutOfBoundsException

PC 1

M,1, L

PC 2

M,1, L
CONCURRENT EDITING: INITIAL STATE (REVIEW)
CONCURRENT INSERTIONS (REVIEW)
Delivering Conflicting Remote Operation (Review)
SYNCHRONOUS RELAYED BROADCAST (REVIEW)

Delay (extra hop) state of operation issue not same as state of execution, though all sites are consistent.
Ordering with Atomic Broadcast (Review)
SECOND OPERATION EXECUTES (REVIEW)

Common state but “intention” violation

Context of operation not the same as when it was issued
ATOMIC BROADCAST ORDERING

PC 1
- D,1
- M,1, L

PC 2
- D,1
- M,1, L
SECOND OPERATION CAUSES EXCEPTION

PC 1
ArrayIndexOutOfBoundsException

PC 2
ArrayIndexOutOfBoundsException

Common behavior but exception

Context of operation not the same as when it was issued
**Single-User Case: Selective Undo**

Undoing non last operation is done in a context different from the one in which it was executed.
Can concurrency control explain or fix this?
Non serializable transactions

If we employed optimistic CC we would abort, but no CC was employed, hence divergent state
VALIDATION/CHECKING TIME (REVIEW)

- Early
  - Pessimistic
- Late
  - Optimistic
- Merging
EARLY VS. LATE VALIDATION (Review)

- Per-operation checking and communication overhead
  - No compression possible.
  - Prevents inconsistency.
  - Tight coupling: incremental results shared
  - Not functional if disconnected
    - Unless we lock very conservatively, limiting concurrency.

- No per-operation checking, communication overhead
  - Compression possible.
  - Inconsistency possible resulting in lost work.
  - Allows parallel development.
  - Functional when disconnected.
MERGING (REVIEW)

- Like optimistic
  - Allow operation to execute without local checks
- But no aborts
  - Merge conflicting operations
  - E.g. insert 1, a || insert 2, b = insert 1, a; insert 3, b || insert 2, b; insert 1, a
- Serializability not guaranteed
  - Ignore reads
  - New transaction to replace conflicting transactions
  - Strange results possible
    - E.g. concurrent dragging of an object in whiteboard
- App-specific
Sometimes semantics can be used to transform concurrent operations to give desired result

Bound to text buffer?
Assume Indexed Sequence Data Type

1. Element 1
2. Element 2
3. Element 3

Arbitrary element type

1. Lunch?
2. Lunch?
3. No, Yes

String element type (message sequence)

1 2 3 4 5
l u n c h

Char element type (text editors)

Insert (index, element)
Delete (index)

Operations
CONCURRENT INORDER INTERACTION

Assume only two sites
Assume messages are received in order from other site
No need for received buffer
Still need time stamp to discover concurrency
General rule for transforming operation

Received Buffer\textsubscript{1}

Received Buffer\textsubscript{2}
**Transformation Function**

- Increment index of remote operation if it has higher index before processing it.
- Remote operation, $R$ transformed!
- Based on index of concurrent local operation, $L$.
- $R^T = \text{Transform} (R, L)$
- Apply $R^T$ instead of $R$ at local site.
- $R$ transformed if $R^T \neq R$.
- Need local buffer to store $L$ rather than remote buffer.

Local Buffer

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,u</td>
<td>5,0</td>
</tr>
<tr>
<td>1,5,?</td>
<td>4,1</td>
</tr>
</tbody>
</table>

PC 1

PC 2
**Control Algorithm: Single Local Concurrent Operation**

Given Remote op R, concurrent with exactly one local op L

$$R^T = \text{Transform } (R, L)$$

Execute $$R^T$$

$$\text{Site.TimeStamp.increment}(R\text{.site})$$
OT SYSTEM COMPONENTS

Transformation function: Handles single local and concurrent operations

Control algorithm: Calls transformation function, processed buffer and local time stamps

Both must be correct.
INCLUSION TRANSFORMATION

Operation Transform (Operation R, Operation L) {
    if (R.type == Insert && L.type == Insert)
        return TransformInsertInsert (R,L);
    else ....
}

Operation TransformInsertInsert (InsertOperation R, InsertOperation L) {
    Operation R^T = R.deepCopy();
    if (R.index > L.index)
        R^T.index = R.index + 1
    return R^T;
}

Transform includes effect of second operand on first operand

Other uses in which first and second operands are not remote and local operations

Names indicate we include effect of earlier executed local operation on later received concurrent remote operation

Called inclusion transformation

Correct?

Correctness criterion?
**CORRECTNESS CRITERION**

Operation Transform (Operation R, Operation L) {
    if (R.type == Insert && L.type == Insert)
        return TransformInsertInsert (R,L);
    else ....
}

Operation TransformInsertInsert (InsertOperation R, InsertOperation L) {
    Operation R^T = R.deepCopy();
    if (R.index > L.index)
        R^T.index = R.index + 1
    return R^T;
}
Initial State

PC 1

PC 2

l u n c h

l u n c h
INSERTION AT SAME INDEX

PC 1

I,6,?

Local Buffer

l u n c h

I,6,

6 0

PC 2

I,6,!

Local Buffer

l u n c h

I,6,

5 1

I,6,!

6 0
**Insertion at same index**

InsertOperation TransformInsertInsert (InsertOperation Remote, InsertOperation Local) {
    Operation Remote\textsuperscript{T} = Remote.clone();
    if (Remote.index > Local.index)
        Remote\textsuperscript{T}.index = Remote.index + 1
    return Remote\textsuperscript{T};
}

Neither operation is transformed
**Insertion at same index Algorithm**

```java
InsertOperation TransformInsertInsert (InsertOperation Remote, InsertOperation Local) {
    Operation RemoteT = Remote.clone();
    if (Remote.index > Local.index)
        RemoteT.index = Remote.index + 1
    return RemoteT;
}
```

**Diagram:**

- **Local Buffer₁:**
  - 1,6,?
  - 6 0

- **Local Buffer₂:**
  - 1,6,!
  - 5 1

**Inconsistency:**
- Neither operation is transformed
**INSERTION AT SAME INDEX: ERROR**

Operation TransformInsertInsert (InsertOperation R, InsertOperation L) {
    Operation $R^T = R$.deepCopy();
    if (R.index > L.index)
        $R^T$.index = R.index + 1
    return $R^T$;
}

Local Buffer

Local Buffer

Constraint for Transform

1 2 3 4 5 6 7
l u n c h ! ?

1 2 3 4 5 6 7
l u n c h ? !
InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L) {
    Operation R^T = deepClone();
    if ((R.index > L.index) ||
        (R.index == L.index && R.id < L.id))
        R^T.index = R.index + 1;
    return R^T;
}

Constraint for Transform

Can two remote operations be transformed wrt to the same local operation?

Insert concurrent text at some position in order of priority
CONTROL ALGORITHM: SINGLE LOCAL CONCURRENT OPERATION

Given Remote op, R, concurrent with exactly one local op L

\[ R^T = \text{Transform} \ (R, L) \]

Execute \( R^T \)

\[ \text{Site.TimeStamp.increment}(R\text{.site}) \]

Local Buffer\(_1\)

Local Buffer\(_2\)
Multiple Transformed Remote Concurrent Operations

A remote site can execute multiple operations that are concurrent wrt to local buffer
**Multiple Transformed Remote Concurrent Operations**

- **Local Buffer\(_1\)**:
  - 1: I,2,u
  - 2: 5
  - 3: 0
  - 4: I,5,
  - 5: I,6!

- **Local Buffer\(_2\)**:
  - 1: I,5,
  - 2: 4
  - 3: 1
  - 4: I,6!
  - 5: 4
  - 6: 2

- **PC1**:
  - 1: I,2,u

- **PC2**:
  - 2: I,5,
  - 3: I,6!
Site 1 Operation Arrives and is Not Transformed
FIRST SITE 2 OPERATION ARRIVES
FIRST OPERATION TRANSFORMED
SECOND OPERATION ARRIVES

Multiple remote operations transformed with respect to same local operation

Local Buffer₁

Local Buffer₂

PC 1

PC 2
**Buffer Cleanup**

Multiple remote operations transformed with respect to same local operation

How long should local buffer be kept?

No need to transform

If local.timestamp < msg.timestamp

Each subsequent message has larger time stamp

Remove all locals from buffer with time stamp smaller than time stamp of received message
Sometimes semantics can be used to transform concurrent operations to give desired result.
OT SYSTEM COMPONENTS

Transformation function: Handles single local and concurrent operations

Control algorithm: Calls transformation function, processed buffer and local time stamps

Both must be correct.
InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L) {
  Operation R^T = deepClone();
  if ((R.index > L.index) ||
       (R.index === L.index && R.id < L.id))
    R^T.index = R.index + 1;
  return R^T;
}
**CONTROL ALGORITHM: SINGLE LOCAL CONCURRENT OPERATION**

Given Remote op $R$, concurrent with exactly one local op $L$

$R^T = \text{Transform} (R, L)$

Execute $R^T$

Site.TimeStamp.increment($R$.site)
Need For Local Buffer

Local Buffer$_1$

Local Buffer$_2$
Multiple remote operations transformed with respect to same local operation

How long should local buffer be kept?

No need to transform

If local.timestamp < msg.timestamp

Each subsequent message has larger time stamp

Remove all locals from buffer with time stamp smaller than time stamp of received message
**DUAL: OPERATION TRANSFORMED MULTIPLE TIMES**

Local Buffer\textsubscript{1}:
- \(I,2, u\)
- \(I,5, ?\)
- \(I,6, !\)

Local Buffer\textsubscript{2}:
- \(I,2, u\)
- \(I,5, ?\)
- \(I,6, !\)

Multiple remote operations transformed with respect to same local operation

Dual?

A remote operation transformed with respect to multiple local operations
**Initial State**

Local Buffer$_1$

Local Buffer$_2$
MULTIPLE CONCURRENT LOCAL OPERATIONS wrt REMOTE CONFLICTING OPERATION

Local Buffer₁

I,4, ? 4 0

Local Buffer₂

I,1,1 3 1
I,2, u 3 2
MULTIPLE CONCURRENT LOCAL OPERATIONS WRT REMOTE CONFLICTING OPERATION

Local Buffer$_1$

Local Buffer$_2$
Multiple Concurrent Local Operations wrt Remote Conflicting Operation

Local Buffer$_1$

I,4, ? 4 0

Local Buffer$_2$

I,1,1 3 1
I,2, u 3 2
Multiple Concurrent Local Operations w.r.t. Remote Conflicting Operation

Local Buffer$_1$

\[
\begin{array}{c}
I,4,? \\
1,4,? \\
I,1,1 \\
I,2,u \\
I,2,u \\
\end{array}
\]

Local Buffer$_2$

\[
\begin{array}{c}
I,1,1 \\
I,1,1 \\
I,2,u \\
I,2,u \\
I,2,u \\
\end{array}
\]
MULTIPLE CONCURRENT LOCAL OPERATIONS wrt REMOTE CONFLICTING OPERATION

Local Buffer₁

Local Buffer₂
MULTIPLE CONCURRENT LOCAL OPERATIONS WRT REMOTE CONFLICTING OPERATION

Local Buffer\textsubscript{1}

\begin{tabular}{|c|c|c|c|}
\hline
I,1,1 & 3 & 0 \\
\hline
I,2, u & 3 & 2 \\
\hline
\end{tabular}

Local Buffer\textsubscript{2}

\begin{tabular}{|c|c|c|c|}
\hline
I,1,1 & 3 & 1 \\
\hline
I,2, u & 3 & 2 \\
\hline
\end{tabular}
**First Transformation**

Transform wrt to first concurrent local operation

Local Buffer

Local Buffer

1 2 3 4 5 6
1 l u n c h ?

1 2 3 4 5
1 l u n c h
SECOND TRANSFORMATION/CONTROL ALGORITHM

Transform wrt to second concurrent local operation

A remote operation transformed with respect to multiple local operations

Run transform function with respect to all concurrent operations in the local log: Transform (Transform (Transform (R, L1), L2) …LN)

Control algorithm now handles multiple concurrent/local operations using Transform function addressing single concurrent remote/local operation

Local Buffer$_1$

Local Buffer$_2$
CONTROL ALGORITHM: SINGLE LOCAL CONCURRENT OPERATION

Given Remote op, R, concurrent with exactly one local op L

\[ R^T = \text{Transform} (R, L) \]

Execute \( R^T \)

\[ \text{Site.TimeStamp.increment}(R.\text{site}) \]

Algorithm for multiple local concurrent operations?
CONTROL ALGORITHM: MULTIPLE CONCURRENT LOCAL OPERATIONS

Given Remote op, R, concurrent with local ops $L^1, L^2, \ldots L^N$

For each $L$

$R = \text{Transform} \ (R, L)$

Execute $R$

$\text{Site.TimeStamp.increment}(R.\text{site})$
All examples so far involved transformation(s) at one site

Transformation at both sites?

Local Buffer$_1$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>I,4,?</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local Buffer$_2$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>I,1,1</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I,2,u</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Initial State**

- **Local Buffer**
  - Local Buffer\(_1\)
  - Local Buffer\(_2\)
MULTIPLE CONCURRENT REMOTE OPERATIONS

Local Buffer₁

1,5, ?
1,5, ? 5 0

Local Buffer₂

I,1,1 4 1
I,6, ! 4 2
ARRIVAL AT SITE 1

Local Buffer$_1$

Local Buffer$_2$

I,5, ? 5 0
I,1,l

I,1,l 4 1
I,6, ! 4 2

I,1,l 4 1
I,6, ! 4 2
RESULT OF TRANSFORM

Local Buffer\textsubscript{1}

\begin{tabular}{cccc}
1 & 2 & 3 & 4 \\
\hline
? & ? & \textcolor{red}{?} & 5 \\
\end{tabular}

\begin{tabular}{cccc}
5 & 0 & \textcolor{red}{0} & 4 \\
\end{tabular}

Local Buffer\textsubscript{2}

\begin{tabular}{cccc}
1 & 2 & 3 & 4 \\
\hline
\textcolor{red}{?} & \textcolor{red}{?} & \textcolor{red}{?} & 5 \\
\end{tabular}

\begin{tabular}{cccc}
4 & 1 & \textcolor{red}{1} & 2 \\
\end{tabular}

Remote operation not transformed
**Untransformed Application**

Local Buffer$_1$

- I,5, ?
- 5
- 0

Local Buffer$_2$

- I,1,1
- 4
- 1
- I,6, !
- 4
- 2

PC$_1$

- 5
- 1

PC$_2$

- 4
- 2
ARRIVAL AT SECOND SITE

Local Buffer₁
- I,5, ? 5 0

Local Buffer₂
- I,1,1 4 1
- I,6, ! 4 2

PC 1
- I,5, ?
- I,1,1

PC 2
- I,1,1
- I,6, !
- I,5, ?

1 2 3 4 5 6
l u n c h ?

1 2 3 4 5 6
l u n c h !
TRANSFORMED wrt TO FIRST LOCAL OPERATION

Local Buffer$_1$

I,5, ?   5   0

Local Buffer$_2$

I,1,1   4   1
I,6, !   4   2

PC 1

PC 2

I,1,1

I,6, ?

I,6, !
EXAMINING SECOND LOCAL OPERATION

Local Buffer$_1$

I,5, ?  5  0

Local Buffer$_2$

I,1,1  4  1
I,6, !  4  2
SECOND TRANSFORMATION AND APPLICATION

Local Buffer\textsubscript{1}

\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
1 & u & n & c & h & ? \\
\hline
\end{tabular}

Local Buffer\textsubscript{2}

\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
1 & I,1, & 4 & 1 & I,6, & 4 & 2 \\
\hline
\end{tabular}

User 1 < User 2
SECOND ARRIVAL AT SITE 1
Operation Transformed

Local Buffer$_1$
- I,5, ?
- 5
- 0

Local Buffer$_2$
- I,1,1
- 4
- 1
- I,6, !
- 4
- 2
**APPLICATION OF TRANSFORMED OPERATION**

Local Buffer<sub>1</sub>
- I,5, ?
- I,1, 1
- I,7, !

Local Buffer<sub>2</sub>
- I,1, 1
- I,6, !
- I,7, ?

PC<sub>1</sub>
- I,5, ?
- I,1, 1
- I,7, !

PC<sub>2</sub>
- I,5, ?
- I,1, 1
- I,6, !

Inconsistency!

Never compared user ids at site 1

What went wrong?
**WHAT WENT WRONG?**

**PC 1**

- Local Buffer 1:
  - I,5, ?
  - 1,2,3,4,5,6,7
  - Lunch ? !
  - I,5, ?  5  0

**PC 2**

- Local Buffer 2:
  - I,1,1
  - I,6, !
  - 1,2,3,4,5,6,7
  - I,1,1  4  1
  - I,6, !  4  2

---

**Never compared site ids at site 1**

**Effect of remote I, 1, 1 on local I, 5, ? not recorded**

**Must change time stamp and if necessary operands of local operation**

**Transform gives effect of an operation on another**

**Used it so far to get effect of local operation on remote operation**

**Need to also use it to determine effect of remote operation on local operation**
Running Transform in Pairs

Transform (Insert (1, ‘l’), Insert (5, ‘?’))

Transform (Insert (5, ‘?’), Insert (1, ‘l’))

Each transformation computed at both sites!

Time stamps of local operations changed

Local Buffer\textsubscript{1}

Local Buffer\textsubscript{2}
Given Remote op, R, concurrent with local ops L₁, L₂, .. L_N

For each L

R = Transform (R, L)

Execute R

Site.TimeStamp.increment(R.site)

Effects of L₁, L₂, .. L_N included in R

Effects of R must also be included in L₁, L₂, .. L_N

Local Buffer₁

Local Buffer₂
NEW CONTROL ALGORITHM

Given Remote op, R, concurrent with local ops L¹, L², .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Effects of L¹, L², .. LN included in R

Effects of R included in L¹, L², .. LN
GEOMETRIC PROOF OF CORRECTNESS

One-step process
Given two sequences of concurrent ops, the transformation merge process can be derived.

Perpendiculars to two non-dashed lines meet at a unique point.

Multiple one-step processes to reach state.

Intermediate state not reached at both sites.

Without undo.

Local op not changed.

Same transformation computed at both sites.

Including effect of multiple locals on a remote at site 1.

Including effect of multiple remotes on a single local at site 2.
**Geometric Proof of Correctness**

Given Remote op, R, concurrent with local ops $L_1, L_2, \ldots L_N$

For each $L$

- $R = \text{Transform} \left( R, L \right)$
- $L = \text{Transform} \left( L, R \right)$
- $L$.TimeStamp.increment($R$.site)
- Execute $R$
- Site.TimeStamp.increment($R$.site)

Including effect of multiple locals on a remote at site 1

Including effect of multiple remotes on a single local at site 2

SW: process site 1 op

SE: process site 2 op
**Re-Run with Transformations**

Local Buffer$_1$

Local Buffer$_2$
**CONCURRENT INTERACTION**

Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

\[ R = \text{Transform} \left( R, L \right) \]

\[ L = \text{Transform} \left( L, R \right) \]

\[ \text{L.TimeStamp.increment}(R.site) \]

Execute R

\[ \text{Site.TimeStamp.increment}(R.site) \]

Local Buffer₁

- I₅, ?
- 5
- 0

Local Buffer₂

- I₆, !
- 4
- 2
**Remote I, 1, L arrives at Site 1**

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)
Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Remote not transformed
Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Local transformed
REMOTE APPLIED

Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)
Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)
**Remote Transformed But Not First Local**

Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

- \( R = \text{Transform} \ (R, L) \)
- \( L = \text{Transform} \ (L, R) \)
- \( L\text{.TimeStamp.increment}(R\text{.site}) \)
- \( \text{Execute } R \)
- \( \text{Site.TimeStamp.increment}(R\text{.site}) \)
**Compared with Second Local (I. 6, !) at Same Location**

Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)
**Transformation and Application**

Given Remote op, $R$, concurrent with local ops $L^1, L^2, .. LN$

For each $L$

$$R = \text{Transform} \ (R, L)$$

$$L = \text{Transform} \ (L, R)$$

$L$.TimeStamp.increment($R$.site)

Execute $R$

$\text{Site}.\text{TimeStamp}.\text{increment}($R$.site)$

Local Buffer$_1$

I,6, ? 5 1

Local Buffer$_2$

I,1,1 5 1
I,6, ! 5 2

PC 1

PC 2

I,5, ?
I,1,1
I,6, !

I,5, ? 5 0
I,1,1 4 1
I,6, ! 4 2

I,7, ?
SECOND REMOTE OPERATION ARRIVES AT SITE 1

Given Remote op, R, concurrent with local ops L¹, L², .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Local Buffer₁
I,6, ? 5 1

Local Buffer₂
I,1,1 5 1
I,6, ! 5 2
Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)
**UNTRANSFORMED REMOTE APPLIED**

Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L:

- \( R = \text{Transform} \ (R, L) \)
- \( L = \text{Transform} \ (L, R) \)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

N Users?
Given Remote op, R, concurrent with local ops $L^1, L^2, \ldots, LN$

For each L

\[ R^T = \text{Transform} \ (R, L) \]

\[ L = \text{Transform} \ (L, R) \]

\[ L.\text{TimeStamp}.\text{increment}(R.\text{site}) \]

\[ R = R^T \]

Execute R

\[ \text{Site.TimeStamp}.\text{increment}(R.\text{site}) \]
3-User Concurrent

$v_1$

PC 1

1 0 0

$v_2$

PC 2

0 1 0

$v_3$

PC 3

0 0 1

Local Buffer

Local Buffer

Local Buffer

1 0 0

1 0 1

0 0 1

I,1, a 1 0 0

I,1, b 0 1 0

I,1, c 0 0 1

I,1, a 1 0 0

I,1, b 0 1 0

I,1, c 0 0 1

I,1, a 1 0 0

I,1, b 0 1 0

I,1, c 0 0 1
TRANSFORM AND APPLY RECEIVED COMMAND

Local Buffer

Local Buffer

Local Buffer

1,1, a 1 0 0
1,1, b 0 1 0
1,1, c 0 0 1
1,1, a 1 0 0
1,1, b 0 1 0
1,1, c 0 0 1

1,1, a 1 0 0
1,1, b 0 1 0
1,1, c 0 0 1

1,1, a 1 0 0
1,1, b 0 1 0
1,1, c 0 0 1
**Transform Local Command**

---

**Local Buffer_1**

\[ \begin{array}{ccc} 
I_1, a & 1 & 0 & 0 
\end{array} \]

**Local Buffer_2**

\[ \begin{array}{ccc} 
I_1, b & 0 & 1 & 0 
\end{array} \]

**Local Buffer_3**

\[ \begin{array}{ccc} 
I_1, c & 1 & 0 & 1 
\end{array} \]
TRANSFORM AND APPLY RECEIVED COMMAND

Local Buffer$_1$

Local Buffer$_2$

Local Buffer$_3$
**Transform Local Command**

- **Local Buffer\textsubscript{1}**
  - \(I,1, a \ 1 \ 0 \ 0\)

- **Local Buffer\textsubscript{2}**
  - \(I,1, b \ 0 \ 1 \ 0\)

- **Local Buffer\textsubscript{3}**
  - \(I,1, c \ 1 \ 1 \ 1\)

- \(v_1: 1 \ 0 \ 0\)

- \(v_2: 0 \ 1 \ 0\)

- \(v_3: 0 \ 0 \ 1\)

Characters in descending id order

But id of 1 and 2 not compared
Different Receive Order

\[ v_1 \]
1 0 0

PC 1

\[ v_2 \]
0 1 0

PC 2

\[ v_3 \]
0 0 1

PC 3

Local Buffer_1
I,1, a 1 0 0

Local Buffer_2
I,1, b 0 1 0

Local Buffer_3
I,1, c 0 0 1
**Transform and Apply Received Command**

Local Buffer\(_1\)

- I,1, a: 1 0 0

Local Buffer\(_2\)

- I,1, b: 0 1 0

Local Buffer\(_3\)

- I,1, c: 0 0 1

**Diagram**

- PC 1: \(v_1\)
  - a

- PC 2: \(v_2\)
  - b

- PC 3
  - c, b

Connections:
- I,1, a: 1 0 0 from a to v1
- I,1, b: 0 1 0 from b to v1
- I,1, c: 0 0 1 from c to v1
- I,2, b: from v2 to c
- I,1, a: 1 0 0 from a to v1
- I,1, b: 0 1 0 from b to v2
- I,1, c: 0 0 1 from c to v2
- I,2, b: from v2 to c
**Transform Local Command**

- **Local Buffer$_1$**: I,1, a 1 0 0
- **Local Buffer$_2$**: I,1, b 0 1 0
- **Local Buffer$_3$**: I,1, c 0 1 1

The diagram illustrates the transformation of commands across local buffers and processors:

1. **PC 1** with command I,1, a 1 0 0 sends to **PC 2** with command I,1, b 0 1 0.
2. **PC 1** receives command I,2, b 0 0 1 from **PC 3**.
3. **PC 2** sends command I,1, c 0 0 1 to **PC 3**.
4. **PC 4** receives command I,2, c 0 0 1 from **PC 1**.

The diagram shows the flow of commands between the processors and local buffers, illustrating the transformation process.
**Transform and Apply Received Command**

$v_1$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

PC 1

$v_2$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

PC 2

$v_3$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

PC 3

Local Buffer$_1$

<table>
<thead>
<tr>
<th></th>
<th>I,1, a</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Local Buffer$_2$

<table>
<thead>
<tr>
<th></th>
<th>I,1, b</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Local Buffer$_3$

|   | I,1, c | 0 | 1 | 1 |
TRANSFORM LOCAL COMMAND

Local Buffer$_1$

I,1, a 1 0 0

Local Buffer$_2$

I,1, a 1 0 0

I,1, b 0 1 0

Local Buffer$_3$

I,1, c 1 1 1
PROBLEM WITH 2 USERS

- Order of concurrent messages influences output.
- Same output not guaranteed at a single site.
- Same output not guaranteed at all sites.
- Problem independent of whether local operation is transformed.
- To understand better, need state transition diagram
**MULTIPLE REMOTE CONCURRENT OPERATIONS**

At Site 1

At Site 2

Distributed Merge

3-User Transition Diagram?
N-Users?
Path 1 for User 1
The two paths must give equivalent results.

In our example, our transformation functions did not!

Necessary condition for new transformation functions?

Each edge should have a unique label.
**Multiple Paths and Edge Labels**

\[ T(T(0_1, 0_2), T(0_3, 0_2)) \]

\[ T(T(0_3, 0_2), T(0_1, 0_2)) \]

\[ T(T(0_3, 0_1), T(0_1, 0_1)) \]

\[ T(T(0_3, 0_1), T(0_2, 0_1)) \]

Constraints for Transform

**TP1**

- \( O_1 \) - \( T(O_2, O_1) \)
- \( O_2 \) - \( T(O_1, O_2) \)

**TP2**

- \( T(T(0_3, 0_2), T(0_1, 0_2)) \)
- \( T(T(0_3, 0_1), T(0_2, 0_1)) \)
Linear local buffer does not suffice, local operation wrt to which a remote operation is transformed depends on received concurrent operations from other sites.

Constraints for Transform:

TP1

\[ O_1 \rightarrow T(O_2, O_1) \]

TP2

\[ T(T(O_3, O_2), T(O_1, O_2)) \]

\[ = \]

\[ T(T(O_3, O_1), T(O_2, O_1)) \]
LINEAR BUFFER VS INTERACTION MODEL (POST LECTURE)

Multiple concurrent paths from a vertex, must store path

T(0, \ O_1)

T(O_3, \ O_1)

O_1 \ no \ longer \ in \ buffer \ to \ compute \ T(O^2, \ O^1)

Local Buffer_1

T(O_1, \ O_3)
**Sufficient Conditions?**

Claim: TP1 and TP2 sufficient for N users

Creating functions meeting TP2 has been problematic

**Claim:** TP1 and TP2 sufficient for N users

**Constraints for Transform**

Google implementation?
Delay (extra hop) state of operation issue not same as state of execution, though all sites are consistent.
ORDERING WITH ATOMIC BROADCAST (REVIEW)
SECOND OPERATION EXECUTES (REVIEW)

PC 1

1 2 3 4 5 6
l u n c ? h

PC 2

1 2 3 4 5 6
l u n c ? h

Common state but “intention” violation

Context of operation not the same as when it was issued
SYNCHRONOUS RELAYED BROADCAST

PC 1

lunch

Relayer

PC 2

I, 2, u

PC 3

lunch

lunch

PC 1

I, 2, u

I, 2, u

I, 2, u
ASYNCHRONOUS MERGED RELAYED BROADCAST

PC 1

lunch

I, 2, u

Relaying Merger

I, 2, u

PC 2

lunch

I, 2, u

PC 3

lunch
2 TO N USERS

- Can do one N-user merge
- Can do N 2-User messages
  - Through a server
  - Each client is consistent with the server
  - Implies each client is consistent with the server
- But server does not issue any operations
  - For each client, server operations are those issued by other clients
REPLICATED ARCHITECTURE WITH CENTRAL MERGER: LOCAL, REMOTE TIME STAMP

Replicated Mapping

UI 1 → PC 1 → Local Buffer 1 → Merger → Local Buffer2 → PC 2 → UI 2

UI 2 → PC 2 → Local Buffer 2 → Merger → Local Buffer3 → PC 3 → UI 3

PC 1 → Local Buffer 1 → Merger → Local Buffer2

PC 2 → Local Buffer 2 → Merger → Local Buffer3

PC 3 → Local Buffer 3

#1, #2 + #3

#1 + #3, #2

#3, #1 + #2

#2, #1 + #3

#1, #2 + #3

#1 + #2, #3

#3

(O^T)^T

OT
**CLIENT AND SERVER RECEIVE ALGORITHM**

Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

\[ R^T = \text{Transform} \left( R, L \right) \]

\[ L = \text{Transform} \left( L \cdot R \right) \]

\[ L.\text{TimeStamp}.\text{increment}(R.\text{site}) \]

\[ R = R^T \]

Execute R

\[ \text{Site.TimeStamp}.\text{increment}(R.\text{site}) \]

---

Given Remote op, R, concurrent with local ops L₁, L₂, .. LN

For each L

\[ R^T = \text{Transform} \left( R, L \right) \]

\[ L = \text{Transform} \left( L \cdot R \right) \]

\[ L.\text{TimeStamp}.\text{increment}(R.\text{site}) \]

\[ R = R^T \]

For all other sites assume server executed R

\[ \text{Site.TimeStamp}.\text{increment}(R.\text{site}) \]
**TRANSFORM OPERATION FOR CLIENT-SERVER CASE**

InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L) {
    Operation $R^T = \text{deepClone}();$
    if ((R.index > L.index) || (R.index == L.index && R.id < L.id))
        $R^T.index = R.index + 1;$
    return $R^T;$
}

InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L) {
    Operation $R^T = \text{deepClone}();$
    if ((R.index > L.index) || (R.index == L.index && !R.isServer()))
        $R^T.index = R.index + 1;$
    return $R^T;$
}
Causality Manager

Causality – Unaware Application

Send Filter → Causality Manager → Receive Filter

Causality-unaware Communication system
CLIENT OT MANAGER

OT - Unaware Application

Send Filter → OT Manager → Receive Filter

OT-unaware Communication system

Single OT Manager?
One for each sequence
SERVER OT MANAGER

OT – Unaware Relayer

Send Filter → OT Manager → Receive Filter

OT-unaware Communication system

Receiver immediately sends, no execution
CLIENT OT MANAGER (REVIEW)

OT – Unaware Application

Send Filter → OT Manager → Receive Filter

OT-unaware Communication system

Single OT Manager?
One for each sequence
**Single Server Filter**

OT – Unaware Relayer

Send Filter ➔ OT Manager ➔ Receive Filter

OT-unaware Communication system

- Receiver immediately sends, no execution
- Can use only one filter
- Receiver does not know identities of destinations but sender may (in current implementation filter called before multicasting)
- Single send filter sufficient
Server OT Managers

OT – Unaware Relayer

Send Filter → OT Manager → OT-unaware Communication system

One OT Manager for each client and sequence

How to attach send filter to server?
**MESSAGE FILTER INTERFACE**

```java
public interface MessageFilter<MessageType> {
    public void setMessageProcessor (MessageProcessor<MessageType> newVal);
    public void filterMessage(MessageType message);
}
```

- **Called by communication system when new message to be filtered available**
- **Next stage in pipeline, processing the filtered message**
- **ReceivedMessage or SentMessage**
- **Called by communication system when pipeline setup**
public interface ServerMessageFilter extends MessageFilter<SentMessage> {
    public void userJoined(String aSessionName, String anApplicationName, String userName);
    public void userLeft(String aSessionName, String anApplicationName, String userName);
}
public interface ServerMessageFilterCreator {
    ServerMessageFilter getServerMessageFilter();
}
public class SentMessageFilterSelector {
    static MessageFilterCreator<SentMessage> filterFactory =
        new AMMessageForwarderCreator<SentMessage>();
    public static MessageFilterCreator<SentMessage> getMessageFilterCreator() {
        return filterFactory;
    }
    public static void setMessageFilterCreator(
        MessageFilterCreator<SentMessage> theFactory) {
        filterFactory = theFactory;
    }
}
SERVER OT MANAGERS

OT – Unaware Relayer

Send Filter

OT Manager

OT-Unaware communication system

One OT Manager for each client and sequence

How to attach send filter to server?
CLIENT OT MANAGER

OT – Unaware Application

Send Filter → OT Manager → Receive Filter

OT-Unaware communication system

Single OT Manager?
One for each sequence
**CLIENT INITIALIZATION**

**Init**

Create <List, OT Manager> Mapping ListOTManager

For each List, L

ListOTManager(L) ← new OT Manager (ClientName, Not Server)

Create Client Send and Receive Filter Factories, passing them OTManager so they can pass them to the two filters
**Send Filter Traceable Steps**

- **Send Filter**
  - On each user edit about OT List L
  - Ask ListOTManager(L) to time stamp edit
  - OTListEditSend the timestamped edit through message processor
  - Ask ListOTManager(L) to store copy of sent message

As in causality must ensure changing site time stamp does not change message time stamp
# Receive Filter Traceable Steps

## Receive Filter

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>On each OTListEditReceived for list L received (through server)</td>
</tr>
<tr>
<td></td>
<td>OTListEditFlipped time stamp</td>
</tr>
<tr>
<td></td>
<td>Ask OTManager(L) to transform received edit</td>
</tr>
<tr>
<td></td>
<td>Pass transformed edit to message processor</td>
</tr>
</tbody>
</table>
OT MANAGER: INIT

Init (User Name, IsServer)

InitialOTTimeStampCreated
OT Manager: Send Steps

1. Time stamp edit
2. LocalSiteCountIncremented
3. Timestamp edit with local time stamp
4. Store Sent Message
5. MessageBuffered
OT MANAGER: RECEIVE FILTER COMMUNICATION

Process received timestamped edit

For each local message not concurrent with received edit

Local MessageUnBuffered

For each local buffered concurrent edit L

L = TransformationResult from Transform(L, R)

R = TransformationResult from Transform(R, L)

OTListEditRemoteCountIncremented in L

RemoteSiteCountIncremented

User name in trace step is name of user who executed the operation, for local edit, the local user, for remote edit, the remote user (exact name, not server)
CLIENT OT MANAGER

OT – Unaware Application

Send Filter  OT Manager  Receive Filter

OT-Unaware communication system
SERVER OT MANAGERS

OT – Unaware Relayer

Send Filter → OT Manager

OT-Unaware communication system

One OT Manager for each client and sequence

May want to extend one list implementation to multiple lists
Could have used this architecture (master/delegate filters) also for clients, but client filters were simple and there were two of them, so it is not clear creating two additional master filters for clients is worth it.
Server Initialization

Init

Create <List, ServerFilter> Mapping ServerFilter

For each List, L

ServerFilter(L) ← new ServerFilter()

Create Send Filter Factory, passing it ServerFilterMapping
public interface ServerMessageFilter extends MessageFilter<SentMessage> {
    public void userJoined(String aSessionName, String anApplicationName, String userName);
    public void userLeft(String aSessionName, String anApplicationName, String userName);
}
MASTER SERVER SEND FILTER: FORWARDING

Join

For each list L
ServerFilter(L).userJoined()

Leave

For each list L
ServerFilter(L).userLeft()

Set Message Processor

For each list L
ServerFilter(L).setMesageProcessor()
New Message

On each client edit of list L

ServerFilter(L).filterMessage()
**Server Send Filter**

**Init**

Create <User, OTManager> Mapping UserOTManager

**Join**

On join of each user U

UserOTManager(U) ← new OTManager(U, Is Server)
<table>
<thead>
<tr>
<th>Server Send Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New Message</strong></td>
</tr>
</tbody>
</table>

| On each OTListEditReceived from Sending User S |
| OTListEditFlipped time stamp                 |
| Ask UserOTManager(S) to create transform received edit, $O^T$ |

| For each user R in UserOTManager other than S |
| Create unicast copy of message containing, $O^{TR}$ |
| By calling ASentMessage.toSpecificUser(message, R) |
| Ask UserOTManager(R) to time stamp $O^{TR}$ |
| Send timestamped edit through message processor |
| Ask ListOTManager(R) to store copy of sent message |
**Centralized Algorithm**

- Assume all users merge through central server
  - Output is produced locally immediately
- Server keeps local buffer and timestamp for each client
- Each client treats server as second user and sends it each command
- Instead of applying (possibly transformed) command to its local state server sends time-stamped command to each remote client
- Client transforms it further if it has executed concurrently
- Client assumes command executed directly by server
- Each client consistent with server, and thus with each other client
- Unique ordering of all commands from remote machines
HISTORY

- dOPT (Distributed Operation Transformation) – Ellis and Gibbs ’89
  - Did not transform local operation
  - Had known problem with multiple users
- adOPTed (Ressel et al ‘96)
  - Transformed local operation
  - Give conditions for N-user replicated merging
- Jupiter (Nichols, Curtis, et al ‘95)
  - Centralized merging
  - Inventors of LiveMeeting
  - Implemented in GoogleWave