Specification and Implementation of Replicated List
— The Jupiter Protocol Revisited

(OPODIS’2018)

Hengfeng Wei, Yu Huang, Jian Lu

Nanjing University

December 17, 2018
The Main Contribution

The Jupiter protocol [Nichols et al., 1995] for replicated list satisfies the weak list specification [Attiya et al., 2016].

---


The Main Contribution

The Jupiter protocol [Nichols et al., 1995]$^a$ for replicated list satisfies the weak list specification [Attiya et al., 2016]$^b$.


This was proposed as a conjecture in a PODC paper [Attiya et al., 2016].
1. Why do we care about replicated list?

2. What is the weak list specification $A$?

3. How does the Jupiter protocol work?

4. How to prove that Jupiter satisfies $A$?
1. Why do we care about replicated list?
Outline

1. Why do we care about replicated list?

2. What is the weak list specification $A_{\text{weak}}$?
Outline

1. Why do we care about replicated list?

2. What is the weak list specification $A_{\text{weak}}$?

3. How does the Jupiter protocol work?
Outline

1. Why do we care about replicated list?
2. What is the weak list specification $A_{\text{weak}}$?
3. How does the Jupiter protocol work?
4. How to prove that Jupiter satisfies $A_{\text{weak}}$?
Replicated List
Replicated Collaborative Text Editing Systems

(a) Google Docs  
(b) Apache Wave  
(c) Wikipedia  
(d) \LaTeX Editor

Replicas are required to respond to user operations immediately. Updates are propagated to other replicas asynchronously.
Replicated list object: to model the core functionality

\[ \text{INS}(a, p) : \text{Insert } a \text{ at position } p. \]

\[ \text{DEL}(p) : \text{Delete the element at position } p. \]

\[ \text{READ} : \text{Return the list.} \]
Weak List Specification
**Definition (Weak List Specification $A_{\text{weak}}$ [Attiya et al., 2016])**

Informally, $A_{\text{weak}}$ requires the ordering between elements that are not deleted to be consistent across the system.

Specify a global property on all states across the system.
We show that $\mathcal{A}_{\text{weak}}$ can be rephrased as

**Definition (Pairwise State Compatibility Property)**

For any pair of list states, there cannot be two elements $a$ and $b$ such that $a$ precedes $b$ in one state but $b$ precedes $a$ in the other.

\[
\begin{align*}
\sigma_0 : ab & \quad \sigma_1 : ba \\
\sigma_0 : ab & \quad \sigma_1 : ba
\end{align*}
\]
Jupiter
Jupiter adopts the **client-server** architecture [Nichols et al., 1995]:

\[(n + 1) \text{ replicas} \triangleq (n) \text{ Client} + (1) \text{ Server}\]
Jupiter adopts the **client-server architecture** [Nichols et al., 1995]:

\[(n + 1) \text{ replicas} \triangleq (n) \text{ Client} + (1) \text{ Server}\]
Challenge: Conflicts caused by concurrent operations
Challenge: Conflicts caused by concurrent operations
(The server is not drawn.)

\[ C_1 \]
efecte

\[ C_2 \]
efecte

Solution: Operational Transformation (OT) [Ellis and Gibbs, 1989]
Challenge: Conflicts caused by concurrent operations (The server is not drawn.)

\[ C_1 \]
\[ C_2 \]
\[ o_1 = \text{Ins}(f, 1) \]
\[ o_2 = \text{DEL}(5) \]

\[ o_1' = \text{Ins}(f, 1) \]
\[ o_2' = \text{DEL}(6) \]
Challenge: Conflicts caused by concurrent operations
(The server is not drawn.)

\[ C_1 \]
\[ C_2 \]

\[ o_1 = \text{Ins}(f, 1) \quad o_2 = \text{Del}(5) \]

\[ o'_2 = \text{Del}(5) \quad o'_1 = \text{Ins}(f, 1) \]
**Challenge:** Conflicts caused by concurrent operations

(The server is not drawn.)

\[ C_1 \quad \text{efecte} \quad C_2 \quad \text{efecte} \]

\[ o_1 = \text{Ins}(f, 1) \quad o_2 = \text{Del}(5) \]

\[ o'_2 = \text{Del}(5) \quad o'_1 = \text{Ins}(f, 1) \]

\[ C_1 \quad \text{efecte} \quad C_2 \quad \text{efecte} \]

\[ o_1 = \text{Ins}(f, 1) \quad o_2 = \text{Del}(5) \]

\[ C_1 \quad \text{efecte} \quad C_2 \quad \text{efecte} \]

**Solution:** Operational Transformation (OT) [Ellis and Gibbs, 1989]
Challenge: Conflicts caused by concurrent operations
(The server is not drawn.)

\[ C_1 \]
\[ \text{efecte} \]
\[ o_1 = \text{Ins}(f, 1) \]
\[ \text{efecte} \]
\[ o_2 = \text{Del}(5) \]
\[ \text{efecte} \]
\[ o_2' = \text{Del}(5) \]
\[ \text{efecte} \]
\[ o_1' = \text{Ins}(f, 1) \]
\[ \text{efecte} \]

\[ C_2 \]
\[ \text{efecte} \]
\[ \text{efecte} \]
\[ \text{efecte} \]
\[ \text{efecte} \]

Solution: Operational Transformation (OT) [Ellis and Gibbs, 1989]
Commutative: \[ \sigma; o_1; o'_2 \equiv \sigma; o_2; o'_1 \]

[Ellis and Gibbs, 1989]
Q: What if replicas diverge by $\geq 2$ steps?
Q : What if replicas diverge by $\geq 2$ steps?
Q: What if replicas diverge by $\geq 2$ steps?
Q: What if replicas diverge by $\geq 2$ steps?
Key Challenge to Solve:

When a replica \( r \) receives an operation \( o \) from another replica redirected by the server, how should \( o \) be transformed?
Key Challenge to Solve:

When a replica $r$ receives an operation $o$ from another replica redirected by the server, how should $o$ be transformed?

Transformed with which operations and in what order?
Key Challenge to Solve:

When a replica \(r\) receives an operation \(o\) from another replica redirected by the server, how should \(o\) be transformed?

Transformed with which operations and in what order?

Key Ideas:

1. With concurrent operations previously executed at \(r\)
2. In the serialization order of operations established at the server
Jupiter uses 2D state spaces [Xu, Sun, and Li, 2014] to manage the procedure of performing OTs [Ellis and Gibbs, 1989].

Edges are labeled with operations.
Jupiter uses 2D state spaces [Xu, Sun, and Li, 2014] to manage the procedure of performing OTs [Ellis and Gibbs, 1989].

Edges are labeled with operations.

**Local Dimension:** For operations generated by the client

**Global Dimension:** For operations generated by others
Each **client** maintains a 2D state space.

The **server** maintains $n (= 3)$ 2D state spaces, one for each client.
Mismatch!

Global property on all replica states specified by $A_{weak}$

Local view each replica maintains in Jupiter
CJupiter (Compact Jupiter)
CJupiter maintains an *n*-ary ordered state space for each replica.

There can be more than two edges coming from the same node.
CJupiter maintains an \textit{n-ary ordered state space} for each replica.

There can be \textit{more than two edges} coming from the same node.

Edges from the same node are \textit{totally ordered} according to the serialization order of associated operations.
Theorem (Equivalence of CJupiter and Jupiter)

Under the same schedule, the behaviors of corresponding replicas in CJupiter and Jupiter are the same.

Schedule: ISSUE, SEND, and RECEIVE of operations

Behavior: A sequence of replica states
Theorem (Equivalence of CJupiter and Jupiter)

Under the same schedule, the behaviors of corresponding replicas in CJupiter and Jupiter are the same.

Schedule: Issue, Send, and Receive of operations
Behavior: A sequence of replica states

Equivalence from the perspectives of both the server and clients.
At the server side:

Proposition ($n \leftrightarrow 1$ (Informal))

The single $n$-ary ordered state space at the server side in CJupiter is a union of $n$ 2D state spaces at the server side in Jupiter.
At the server side:

Proposition ($n \leftrightarrow 1$ (Informal))

The single $n$-ary ordered state space at the server side in CJupiter is a union of $n$ 2D state spaces at the server side in Jupiter.

At the client side:

Proposition ($1 \leftrightarrow 1$ (Informal))

Jupiter is slightly optimized in implementation at clients by eliminating redundant OTs in CJupiter.
Proposition (Compactness of CJupiter (Informal))

At a high level, CJupiter maintains only one n-ary ordered state space.
Proposition (Compactness of CJupiter (Informal))

At a high level, CJupiter maintains only one n-ary ordered state space.

All replica states are represented in a single data structure.
Proposition (Compactness of CJupiter (Informal))

At a high level, CJupiter maintains only one n-ary ordered state space.

All replica states are represented in a single data structure.

Each replica behavior corresponds to a path going through this state space.
CJupiter Satisfies the Weak List Specification
We focus on a single $n$-ary ordered state space.

We show the **pairwise state compatibility** property in three steps. 

**By Contradiction, By Induction, and By Case Analysis.**
1. Take any two nodes/states $v_1$ and $v_2$.

Lemma (LCA (Lowest Common Ancestor))

Each pair of states in the $n$-ary ordered state space has a unique LCA.

$v_0 = \text{LCA}(v_1, v_2)$
Consider the paths to \( v_1 \) and \( v_2 \) from their LCA \( v_0 \). 

**Lemma (Disjoint Paths)**

The set of operations \( O_{v_0 \rightarrow v_1} \) along \( P_{v_0 \rightarrow v_1} \) is disjoint from the set of operations \( O_{v_0 \rightarrow v_2} \) along \( P_{v_0 \rightarrow v_2} \).

\[ v_0 = \text{LCA}(v_1, v_2) \]
3) Consider the states in these two paths.

**Lemma (Compatible Paths)**

*Each pair of states consisting of one state $v$ in $P_{v_0 \rightarrow v_1}$ and the other $v'$ in $P_{v_0 \rightarrow v_2}$ are compatible.*

$v_0 = \text{LCA}(v_1, v_2)$

In particular, $v_1$ and $v_2$ are compatible.
The main contribution

The Jupiter protocol [Nichols et al., 1995]$^a$ for replicated list satisfies the weak list specification [Attiya et al., 2016]$^b$.

---


This was proposed as a conjecture in a PODC paper [Attiya et al., 2016].
Model checking/verifying a family of Jupiter protocols using TLA+/TLAPS

AbsJupiter (Wei@TR’18)

CJupiter (Wei@OPODIS’18)

XJupiter (Xu@CSCW’14)

AJupiter (Attiya@PersonalComm’17)
Thank You!


Backup
Does Jupiter satisfy the weak list specification?

Yes, it does.
Replication (for availability)

Replicas respond to user operations **immediately**

Updates are propagated **asynchronously**
Definition (Eventual Convergence [Ellis and Gibbs, 1989])
The lists are identical at all replicas at quiescence, i.e., all update operations have been executed at all replicas.
Definition (Eventual Convergence [Ellis and Gibbs, 1989])
The lists are identical at all replicas at quiescence, i.e., all update operations have been executed at all replicas.

Definition (Strong Eventual Consistency [Shapiro et al., 2011])
The lists are identical at all replicas whenever after executing the same set of update operations.
Definition (Eventual Convergence [Ellis and Gibbs, 1989])
The lists are identical at all replicas at quiescence, i.e., all update operations have been executed at all replicas.

Definition (Strong Eventual Consistency [Shapiro et al., 2011])
The lists are identical at all replicas whenever after executing the same set of update operations.

Specify little on intermediate states going through by replicas.
Strong/weak list specification [Attiya et al., 2016]

Specify global properties on all states across the system.

**Specification and Complexity of Collaborative Text Editing**

<table>
<thead>
<tr>
<th>Hagit Attiya</th>
<th>Sebastian Burckhardt</th>
<th>Alexey Gotsman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technion</td>
<td>Microsoft Research</td>
<td>IMDEA Software Institute</td>
</tr>
<tr>
<td>Adam Morrison</td>
<td>Hongseok Yang</td>
<td>Marek Zawirski*</td>
</tr>
<tr>
<td>Technion</td>
<td>University of Oxford</td>
<td>Inria &amp; Sorbonne Universités, UPMC Univ Paris 06, LIP6</td>
</tr>
</tbody>
</table>

**Proved:** RGA [Roh et al., 2011] satisfies the strong list specification.

**Conjecture:** Jupiter [Nichols et al., 1995] satisfies the weak list specification.
It is still challenging to achieve convergence despite the server.

Serializability may not be desirable.

It does not imply that clients process operations in the same order.
\[ \forall \sigma, \sigma' : a, b \in \sigma \cap \sigma' \implies (a \prec_\sigma b \iff a \prec_{\sigma'} b) \]

\((\sigma, \sigma' : \text{list}; \ a, b : \text{element}; \ \prec_\sigma : \text{precedes})\)

\[
\begin{array}{c}
\sigma_0 : ab \\
\sigma_1 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ax \\
\sigma_1 : xb \\
\sigma_2 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ab \\
\sigma_1 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ax \\
\sigma_1 : xb \\
\sigma_2 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ab \\
\sigma_1 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ax \\
\sigma_1 : xb \\
\sigma_2 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ab \\
\sigma_1 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ax \\
\sigma_1 : xb \\
\sigma_2 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ab \\
\sigma_1 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ax \\
\sigma_1 : xb \\
\sigma_2 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ab \\
\sigma_1 : ba \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_0 : ax \\
\sigma_1 : xb \\
\sigma_2 : ba \\
\end{array}
\]
OT functions for a replicated list object [Ellis and Gibbs, 1989]

\[
OT\left(\text{INS}(a_1, p_1, pr_1), \text{INS}(a_2, p_2, pr_2)\right) = \begin{cases} 
\text{INS}(a_1, p_1, pr_1) & p_1 < p_2 \\
\text{INS}(a_1, p_1 + 1, pr_1) & p_1 > p_2 \\
\text{NOP} & p_1 = p_2 \wedge a_1 = a_2 \\
\text{INS}(a_1, p_1 + 1, pr_1) & p_1 = p_2 \wedge a_1 \neq a_2 \wedge pr_1 > pr_2 \\
\text{INS}(a_1, p_1, pr_1) & p_1 = p_2 \wedge a_1 \neq a_2 \wedge pr_1 \leq pr_2 
\end{cases}
\]

\[
OT\left(\text{INS}(a_1, p_1, pr_1), \text{DEL}(\_, p_2, pr_2)\right) = \begin{cases} 
\text{INS}(a_1, p_1, pr_1) & p_1 \leq p_2 \\
\text{INS}(a_1, p_1 - 1, pr_1) & p_1 > p_2 
\end{cases}
\]

\[
OT\left(\text{DEL}(\_, p_1, pr_1), \text{INS}(a_2, p_2, pr_2)\right) = \begin{cases} 
\text{DEL}(\_, p_1, pr_1) & p_1 < p_2 \\
\text{DEL}(\_, p_1 + 1, pr_1) & p_1 \geq p_2 
\end{cases}
\]

\[
OT\left(\text{DEL}(\_, p_1, pr_1), \text{DEL}(\_, p_2, pr_2)\right) = \begin{cases} 
\text{DEL}(\_, p_1, pr_1) & p_1 < p_2 \\
\text{DEL}(\_, p_1 - 1, pr_1) & p_1 > p_2 \\
\text{NOP} & p_1 = p_2 
\end{cases}
\]
Consider a replicated system with \( n (= 3) \) clients.