
Hengfeng Wei  
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China  
hfwei@nju.edu.cn

Yu Huang  
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China  
yuhuang@nju.edu.cn

Jian Lu  
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China  
lj@nju.edu.cn

ABSTRACT

The replicated list object is frequently used to model the core functionality of replicated collaborative text editing systems. Recently, Attiya et al. [2] proposed the strong/weak list specification and conjectured that the well-known Jupiter protocol satisfies the weak list specification. The major obstacle to proving this conjecture is the mismatch between the global property on all replica states prescribed by the specification and the local view each replica maintains in Jupiter using data structures like 1D buffer or 2D state space. To address this issue, we propose CJupiter (Compact Jupiter) based on a novel data structure called n-ary ordered state space for a replicated client/server system with n clients. At a high level, CJupiter maintains only a single n-ary ordered state space which encompasses exactly all states of each replica. We prove that CJupiter and Jupiter are equivalent and that CJupiter satisfies the weak list specification, thus solving the conjecture above.

KEYWORDS

Collaborative text editing systems; Replicated list; Strong/weak list specification; Operational transformation; Jupiter protocol.

ACM Reference Format:


1 INTRODUCTION

Collaborative text editing systems, like Google Docs [5] and Apache Wave [1], allow multiple users to concurrently edit the same document. For availability, such systems often replicate the document at several replicas. For low latency, replicas are required to respond to user operations immediately without any communication with others and updates are propagated asynchronously.

The replicated list object has been frequently used to model the core functionality (e.g., insertion and deletion) of collaborative text editing systems [2, 7, 8]. Recently, Attiya et al. [2] proposed the strong/weak list specification, which specifies global properties on intermediate states going through by replicas. It is conjectured [2] that the Jupiter protocol [7, 8], which is behind Google Docs [5] and Apache Wave [4], satisfies the weak list specification. Jupiter adopts a centralized server replica for propagating updates. 1 Client replicas are connected to the server replica via FIFO channels (Figure 1). Jupiter relies on the technique of operational transformations (OT) to achieve convergence [7]. The basic idea of OT is for each replica to execute any local operation immediately and to transform a remote operation so that it takes into account the concurrent operations previously executed at the replica.

The major obstacle to proving that Jupiter satisfies the weak list specification is the mismatch between the global property on all states prescribed by such a specification and the local view each replica maintains in the protocol. On the one hand, the weak list specification requires that states across the system are pairwise compatible. That is, 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. On the other hand, Jupiter uses data structures like 1D buffer or 2D state space [8] which are not “compact” enough to capture all replica states in one. The replica states are dispersed in multiple data structures maintained locally at individual replicas.

To resolve the mismatch, we propose CJupiter (Compact Jupiter), which uses a novel data structure called n-ary ordered state space for a system with n clients. CJupiter is compact in the sense that at a high level, it maintains only a single n-ary ordered state space which encompasses exactly all states of each replica. Each replica behavior corresponds to a path going through this state space. This makes it feasible for us to reason about global properties and finally prove that Jupiter satisfies the weak list specification, thus solving the conjecture of Attiya et al. The roadmap is as follows:

- (Section 3) We propose CJupiter based on the n-ary ordered state space structure.
- (Section 4) We prove that CJupiter is equivalent to Jupiter in the sense that the behaviors of their corresponding replicas are the same under the same schedule of operations.
- (Section 5) We prove that CJupiter satisfies the weak list specification. Due to the “compactness” of CJupiter, we are able to focus on a single n-ary ordered state space which provides a global view of all possible replica states.

1 Since replicas are required to respond to user operations immediately, the client/server architecture does not imply that clients process operations in the same order.
Figure 1: A schedule of four operations adapted from [2], involving a server replica and three client replicas. The lists produced by CJupiter (Section 3) are shown in boxes.

2 PRELIMINARIES

We follow the framework proposed by Burckhardt et al. [3] for specifying replicated data types. A replica is a state machine $R$, driven by three kinds of events: 1) do $(o, v)$; accepts an operation $o$ from a user and immediately returns a value $v$; 2) sends($m$); sends a message $m$ to some replicas; 3) receive($m$); receives a message $m$. A protocol is a collection $R$ of replicas. An execution $a$ of a protocol $R$ is a sequence of all events occurring at the replicas in $R$. We denote by $e_{\alpha}$ the replica at which an event $e$ occurs, and by $e \prec_{\alpha} e'$ that $e$ precedes $e'$ in $a$. The causally-happens-before relation $\prec_{\alpha}$ on the events in $\alpha$ is defined as usual [6]. The behavior of a replica $R$ in $\alpha$ is a sequence of the form: $\alpha = o_0, e_1, e_2, \ldots$, where $(e_1, e_2, \ldots)$ is the sequence of events at $R$ and for all $i$, $e_i$ is the state obtained by applying $e_i$ on $e_{\alpha-1}$. A state $\sigma$ of $R$ in $\alpha$ is represented by the events in a prefix of $a_{\alpha}$ that has processed.

An abstract execution records the operations performed by users and visibility relationships between them [2]. Formally, it is a pair $A = (H, \text{vis})$, where $H$ is a sequence of do events and $\text{vis} \subseteq H \times H$ is an acyclic visibility relation such that 1) if $e_1 \prec_H e_2$ and $R(e_1) = R(e_2)$, then $e_1 \leadsto e_2$; 2) if $e_1 \leadsto e_2$, then $e_1 \prec_H e_2$; 3) is transitive. A specification $S$ is a prefix-closed set of abstract executions. A protocol $R$ satisfies a specification $S$, denoted $R \models S$, if any execution $a$ of $R$ complies with some abstract execution $A = (H, \text{vis})$ in $S$, namely $\forall R \in R \models H[|R| = a|_R]$, where $a|_R$ is the sequence of do events of replica $R$ in $\alpha$.

A replicated list object supports three types of user operations [2]:

- **Ins($a$, $p$)**: inserts a unique element $a$ at position $p$ and returns the updated list.
- **Del($p$)**: deletes an element at position $p$ and returns the updated list. The parameter $a$ is to record the deleted element.
- **Read**: returns the contents of the list.

We denote by $\text{elems}(A) = \{ a | \text{do}(\text{Ins}(a, \_), \_) \in H \}$ the set of all elements inserted into the list in an abstract execution $A = (H, \text{vis})$.

The weak list specification requires the ordering between elements that are not deleted to be consistent across the system [2].

Definition 1. An abstract execution $A = (H, \text{vis})$ belongs to the weak list specification $\mathcal{A}_{\text{weak}}$ if there is a relation $\text{lo} \subseteq \text{elems}(A) \times \text{elems}(A)$, called the list order, such that:

1. Each event $e = \text{do}(o, w) \in H$ returns a sequence of elements $w = a_0 \ldots a_{n-1}$, where $a_i \in \text{elems}(A)$, such that:
   a. $w$ contains exactly the elements visible to $e$ that have been inserted, but not deleted.
   b. $\text{lo}$ is consistent with the order of the elements in $w$.
   c. Elements are inserted at the specified position: $o = \text{Ins}(a, k) \implies a = a_{\min\{k, n-1\}}$.
2. $\text{lo}$ is irreflexive and for all events $e = \text{do}(o, w) \in H$, it is transitive and total on $\{ a \mid a \in w \}$.

Example 2. In the execution of Figure 1, there exist three states with lists $w_1 = ba$, $w_2 = ax$, and $w_3 = xb$, respectively. This is allowed by the weak list specification with the list order $\text{lo}$: $b \text{lo} a = a \text{lo} x$ on $w_1$, $a \text{lo} x$ on $w_2$, and $x \text{lo} b$ on $w_3$.

3 THE CJUPITER AND JUPITER PROTOCOLS

CJupiter adopts a client/server architecture, where the server serializes operations and propagates them from one client to others. For a client/server system with $n$ clients, CJupiter maintains $(n+1)$ $n$-ary ordered state spaces, one per replica (CSS$_i$ for the server and CSS$_j$ for client $c_j$). Each CSS is a directed graph whose vertices represent states and edges are labeled with operations of type $Op$. Each operation of type $Op$ consists of a globally unique operation identifier $oid$ and an operation context $ctx$ which is a set of $oids$, denoting the operations it "knows". The state in a vertex is represented by the set $oids$ of operations that have been executed. The edges from a vertex are totally ordered by the operations associated with them, which are in turn totally ordered by the server.

Each replica keeps the most recent vertex $cur$ of its $n$-ary ordered state space $S$. We briefly describe CJupiter in three parts.

**Local Processing.** When a client receives a list operation $o$ from a user, it applies $o$ locally, generates $op \in Op$ with the operation context $S.cur.oids$, creates a vertex $v$ with $v.oids = S.cur.oids \cup \{op.oid\}$, links $v$ to $S.cur$ via an edge labeled with $op$, and finally sends $op$ to the server.

**Server Processing.** When the server receives an operation $op \in Op$ from a client, it transforms $op$ with an operation sequence in $S$ to obtain $op'$ by calling $S.XForm(op)$ (see below), applies $op'$ locally, and then sends $op$ (not $op'$) to other clients.

**Remote Processing.** When a client receives an operation $op \in Op$ from the server, it transforms $op$ with an operation sequence in $S$ to obtain $op'$ by calling $S.XForm(op)$, and applies $op'$ locally.

**OTs in CJupiter.** The procedure $S.XForm(op : Op)$ first locates the vertex $u$ whose $oids$ matches the $ctx$ of $op$, and then iteratively transforms $op$ with an operation sequence consisting of operations along the first edges from $u$ to the final vertex $cur$ of $S$. The choice of the "first" edges in OTs is necessary for establishing the equivalence of CJupiter and Jupiter at the server side.

Although $(n+1)$ $n$-ary ordered state spaces are maintained by CJupiter for a system with $n$ clients, they are all the same (Figure 2).
We prove that CJupiter is equivalent to Jupiter from perspectives of Theorem 4. Under the same schedule, the behaviors of corresponding replicas in CJupiter for each replica under the schedule of Theorem 4. The following theorem, together with Theorem 4, solves the conjecture of Attiya et al. [2].

**Theorem 5.** CJupiter satisfies the weak list specification $A_{\text{weak}}$.

**Proof.** For each execution $a$ of CJupiter, we first construct an abstract execution $A = (H, \text{vis})$ with $\text{vis} = h_{\text{seq}}$. It is easy to prove the conditions 1(a) and 1(c) of $A_{\text{weak}}$. Then, we define the list order relation $\text{lo}$: For $a, b \in \text{elems}(A)$, $a \overset{\text{lo}}{\rightarrow} b$ iff there exists an event $e \in a$ with returned list $w$ such that $a$ precedes $b$ in $w$. By definition, $\text{lo}$ satisfies conditions 1(b).

It remains to show the irreflexivity of $\text{lo}$, which is equivalent to the pairwise state compatibility property: $\text{lo}$ is irreflexive iff any two list states $w_1$ and $w_2$ in $A$ are compatible, namely, for any two common elements $a$ and $b$ of $w_1$ and $w_2$, their relative orderings are the same in $w_1$ and $w_2$. By Proposition 3, it suffices to show that the state space $CSS_2$ at the server satisfies the pairwise state compatibility property. Given a pair of states/vertices in $CSS_2$, we consider the paths to them from their LCA. 

**Lemma 6.** Every pair of vertices in $CSS_2$ has a unique LCA.

**Lemma 7.** Let $v_0$ be the unique LCA of a pair of vertices $v_1$ and $v_2$ in $CSS_2$. Then, the path $P_{v_0 \rightarrow v_1}$ from $v_0$ to $v_1$, as well as $P_{v_0 \rightarrow v_2}$ from $v_0$ to $v_2$, is simple, namely, there are no duplicate operations along it. Furthermore, 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 $P_{v_0 \rightarrow v_2}$.

The desired pairwise state compatibility property follows when we take the common vertex $v_0$ in the next Lemma as the LCA of the two vertices $v_1$ and $v_2$ under consideration.

**Lemma 8.** Let $P_{v_0 \rightarrow v_1}$ and $P_{v_0 \rightarrow v_2}$ be two paths from vertex $v_0$ to vertices $v_1$ and $v_2$, respectively in $CSS_2$. If they are disjoint simple paths, then the list states of $v_1$ and $v_2$ are compatible. 

\[\square\]

### 6 ACKNOWLEDGMENTS

This work is supported by the National 973 Program of China (2015CB352202), the National Natural Science Foundation of China (61702253), and the Fundamental Research Funds for the Central Universities (20214380031).

### REFERENCES


---

\[\text{The LCA (Lowest Common Ancestor) of two vertices } v_1 \text{ and } v_2 \text{ in } CSS_2, \text{ which is a DAG}, \text{ is the lowest (i.e., deepest) vertex that has both } v_1 \text{ and } v_2 \text{ as descendants.} \]