Brief Announcement: Specification and Implementation of Replicated List: The Jupiter Protocol Revisited*

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. 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:
Hengfeng Wei, Yu Huang, and Jian Lu. 2018. Brief Announcement: Specification and Implementation of Replicated List: The Jupiter Protocol Revisited. In *PODC '18: ACM Symposium on Principles of Distributed Computing, July 23–27, 2018, Egham, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3212734.3212778

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 data 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.

*The full version is available at https://arxiv.org/abs/1708.04754 (v3, 12 May 2018).

PODC '18, July 23–27, 2018, Egham, United Kingdom

ACM ISBN 978-1-4503-5795-1/18/07.
https://doi.org/10.1145/3212734.3212778

[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.

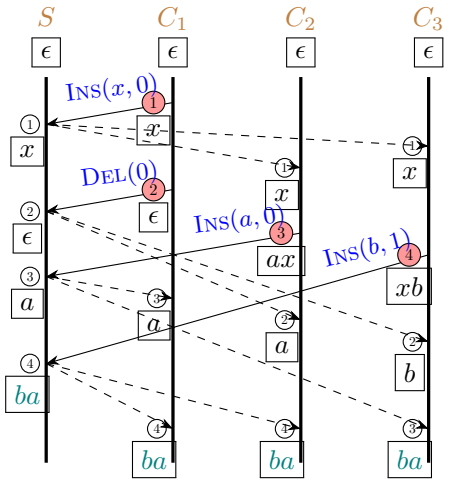

Figure 1: A schedule of four operations adapted from [2], involving a server replica and three client replicas. The lists produced by CJupiter (Section 3) are shown in boxes.

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)* $\mathrm{do}(o, v)$: accepts an operation o from a user and immediately returns a value v; *2)* $\mathrm{send}(m)$: sends a message m to some replicas; *3)* $\mathrm{receive}(m)$: receives a message m. A ***protocol*** is a collection $\mathcal{R}$ of replicas. An ***execution*** α of a protocol $\mathcal{R}$ is a sequence of all events occurring at the replicas in $\mathcal{R}$. We denote by $R(e)$ the replica at which an event e occurs, and by $e \prec_\alpha e'$ that e precedes e' in α. The causally happens-before relation $\xrightarrow{\mathrm{hb}_\alpha}$ on the events in α is defined as usual [6]. The ***behavior*** of a replica R in α is a sequence of the form: $\sigma_0, e_1, \sigma_1, e_2, \ldots$, where $\langle e_1, e_2, \ldots \rangle \triangleq \alpha|_R$ is the subsequence of events at R and for all i, σ_i is the state obtained by applying e_i on σ_{i-1}. A ***state*** σ of R in α is represented by the events in a prefix of $\alpha|_R$ it has processed.

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

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

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

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

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 $A = (H, \mathrm{vis})$ belongs to the ***weak list specification*** $\mathcal{A}_{\mathrm{weak}}$ iff there is a relation $\mathrm{lo} \subseteq \mathrm{elems}(A) \times \mathrm{elems}(A)$, called the *list order*, such that:

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

(2) lo is irreflexive and for all events $e = \mathrm{do}(op, w) \in H$, 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 lo: $b \xrightarrow{\mathrm{lo}} a$ on w_1, $a \xrightarrow{\mathrm{lo}} x$ on w_2, and $x \xrightarrow{\mathrm{lo}} b$ 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_s for the server and CSS_{c_i} for client c_i). 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 *oid*s, 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 $S.\textsc{xForm}(op)$ (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 $S.\textsc{xForm}(op)$, and applies op' locally.

OTs in CJupiter. The procedure $S.\textsc{xForm}(op : Op)$ 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).

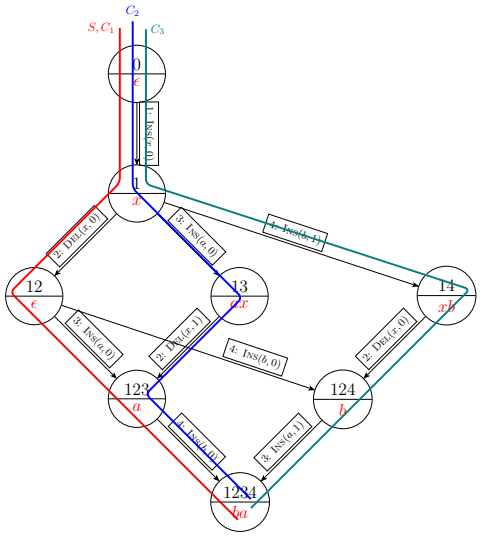

Figure 2: The same final n-ary ordered state space constructed by CJupiter for each replica under the schedule of Figure 1. Each replica behavior corresponds to a path going through this state space.

Proposition 3. *In* CJupiter*, the replicas that have processed the same set of operations have the same n-ary ordered state space.*

Jupiter is similar to CJupiter with three major differences: *1)* For a client/server system with n clients, Jupiter [8] maintains $2n$ ***2D state spaces***, each consisting of a *local* dimension and a *global* dimension. In particular, the server maintains n 2D state spaces, one for each client; *2)* In xForm($op : Op, d \in \{LOCAL, GLOBAL\}$) of Jupiter, the operation sequence with which op transforms is determined by the parameter d; *3)* In Jupiter, the server propagates the *transformed* operations to other clients.

4 CJUPITER IS EQUIVALENT TO JUPITER

We prove that CJupiter is equivalent to Jupiter from perspectives of both the server and clients. ***At the server***, the n-ary ordered state space CSS_s of CJupiter equals the *union* (in terms of graphs as sets of vertices and edges) of all 2D state spaces maintained at the server for each client in Jupiter. The equivalence of ***clients*** follows since the final transformed operations executed at each client in Jupiter and CJupiter are the same (although the original operations are propagated in CJupiter). Thus, we have that

Theorem 4. *Under the same schedule, the behaviors of corresponding replicas in* CJupiter *and* Jupiter *are the same.*

5 CJUPITER SATISFIES THE WEAK LIST SPECIFICATION

The following theorem, together with Theorem 4, solves the conjecture of Attiya et al. [2].

Theorem 5. CJupiter *satisfies the weak list specification* $\mathcal{A}_{weak}$.

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

It remains to show the *irreflexivity* of lo, which is equivalent to the ***pairwise state compatibility property***: 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_s at the server satisfies the pairwise state compatibility property. Given a pair of states/vertices in CSS_s, we consider the paths to them from their LCA. [2]

Lemma 6. *Every pair of vertices in CSS_s 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_s. Then, the path $P_{v_0 \rightsquigarrow v_1}$ from v_0 to v_1, as well as $P_{v_0 \rightsquigarrow v_2}$ from v_0 to v_2, is **simple**, namely, there are no duplicate operations along it. Furthermore, the set of operations $O_{v_0 \rightsquigarrow v_1}$ along $P_{v_0 \rightsquigarrow v_1}$ is **disjoint** from the set of operations $O_{v_0 \rightsquigarrow v_2}$ along $P_{v_0 \rightsquigarrow v_2}$.*

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

□

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 (020214380031).

REFERENCES

[1] [n. d.]. Apache Wave. https://incubator.apache.org/wave/.
[2] Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, and Marek Zawirski. 2016. Specification and complexity of collaborative text editing. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC '16)*. ACM, 259–268.
[3] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. 2014. Replicated Data Types: Specification, Verification, Optimality. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, 271–284.
[4] The Apache Wave Foundation. 2015. *Apache Wave (incubating) Protocol Documentation (Release 0.4)*.
[5] Google. 2010. What's different about the new Google Docs: Making collaboration fast. https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html.
[6] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (July 1978), 558–565.
[7] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. 1995. High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology (UIST '95)*. ACM, 111–120.
[8] Yi Xu, Chengzheng Sun, and Mo Li. 2014. Achieving Convergence in Operational Transformation: Conditions, Mechanisms and Systems. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '14)*. ACM, 505–518.

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