Design and Verification of Byzantine Fault Tolerant CRDTs

Liangrun Da

School of Computation, Information, and Technology
Technical University of Munich

November 15, 2024

Decentralized Collaboration

Wikipedia as an Example

- Wikipedia is currently a centralized system and needs funding to maintain the server.
- A decentralized alternative can be run by volunteers without any central server, hence no funding is needed.
- However, decentralized collaboration systems face a major problem: no consistency guarantee.

What are Conflict-free Replicated Data Types?

RGA as an Example

Hllo

p

Hllo

r

Hllo

q

What are Conflict-free Replicated Data Types?

RGA as an Example

Hello

p

⟨Insert "e" after "H" ⟩

Hllo

r

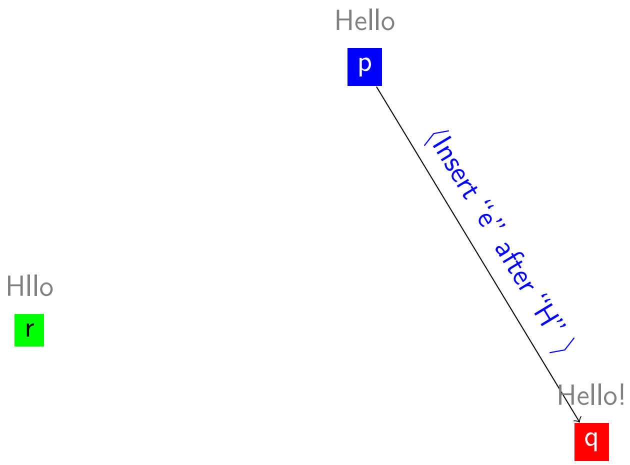
Hllo!

q

⟨Insert "!" after "o" ⟩

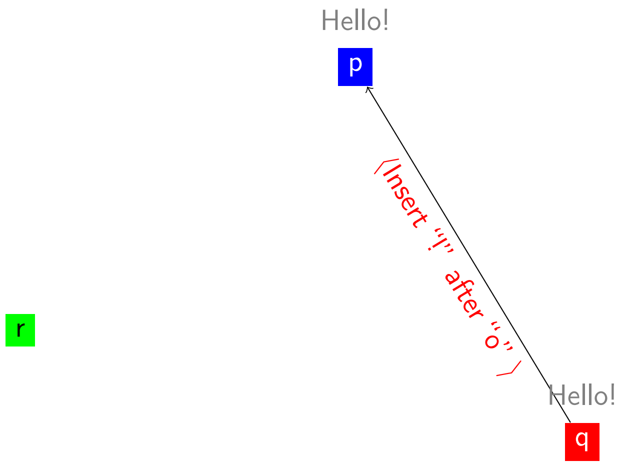
What are Conflict-free Replicated Data Types?

RGA as an Example



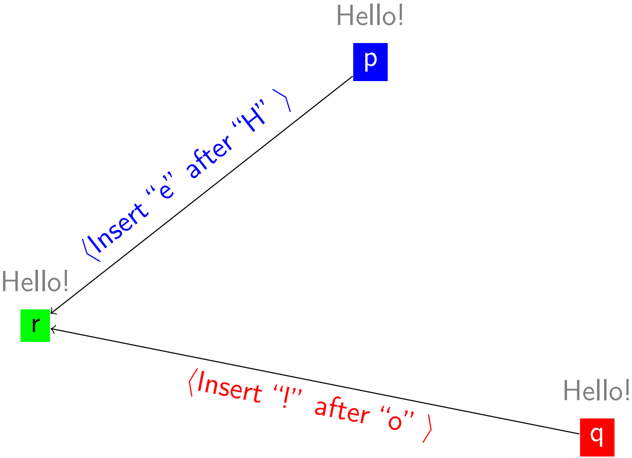
What are Conflict-free Replicated Data Types?

RGA as an Example



What are Conflict-free Replicated Data Types?

RGA as an Example



Strong Eventual Consistency[1]

- Eventual Delivery: an update delivered at some correct node is eventually delivered at all correct nodes
- Convergence: correct replicas that have delivered the same set of updates have equivalent state.
  - Operations are delivered in causal order.
  - Concurrent operations commute.
- Termination: All method executions terminate.

Termination is usually easy to guarantee, so we will focus on the other two.

[1] M Shapiro et al. "Conflict-free replicated data types". In: *Stabilization, Safety, and Security of Distributed Systems: 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings 13*. Springer. 2011, pp. 386–400.

Motivation

- CRDTs are a solution for replication in peer-to-peer systems because they don't require a central server.

Motivation

- CRDTs are a solution for replication in peer-to-peer systems because they don't require a central server.
- Most CRDT algorithms assume participating peers strictly follow the protocol, i.e. they are not Byzantine fault tolerant (BFT).

Motivation

- CRDTs are a solution for replication in peer-to-peer systems because they don't require a central server.
- Most CRDT algorithms assume participating peers strictly follow the protocol, i.e. they are not Byzantine fault tolerant (BFT).
- However, open peer-to-peer systems allow anyone to join or leave, hence the assumption is not safe.

BFT CRDTs vs BFT Consensus

- BFT Consensus requires a total order broadcast, which is unnecessary overhead for CRDTs.

BFT CRDTs vs BFT Consensus

- BFT Consensus requires a total order broadcast, which is unnecessary overhead for CRDTs.
- BFT Consensus provably requires the Byzantine peers to be less than 1/3 of the total peers, where as BFT CRDTs should be able to tolerate any number of Byzantine peers.

BFT CRDTs vs BFT Consensus

- BFT Consensus requires a total order broadcast, which is unnecessary overhead for CRDTs.
- BFT Consensus provably requires the Byzantine peers to be less than 1/3 of the total peers, where as BFT CRDTs should be able to tolerate any number of Byzantine peers.
- BFT Consensus and BFT CRDTs are fundementally different.

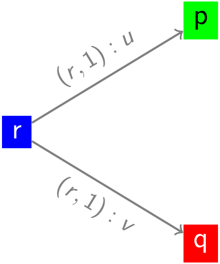
Eventual Delivery

Version vectors are not safe

Eventual Delivery

Version vectors are not safe

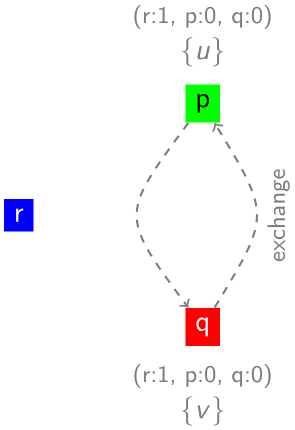
- Peer A sends two different updates with the same version vector.



Eventual Delivery

Version vectors are not safe

- Peer A sends two different updates with the same version vector.
- Peer B and C exchange their version vectors and believe that their updates are the same.



Eventual Delivery

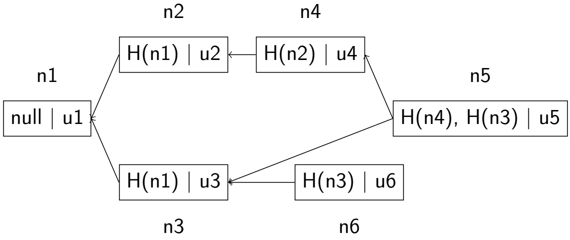
A Naive BFT Protocol

- If a peer sends the set of all updates to other peers, and vice versa, the eventual delivery is guaranteed.
- However it is highly inefficient.

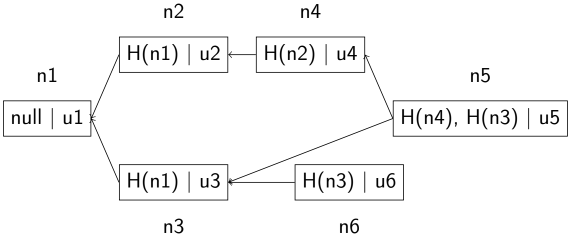
Eventual Delivery

Hash DAG for BFT Eventual Delivery

- Assuming a cryptographic hash function, a hash DAG is a directed acyclic graph where each node has a value (i.e. an update in our context) and a set of hashes, which resolve to its predecessors. (like Git)



- Assuming a cryptographic hash function, a hash DAG is a directed acyclic graph where each node has a value (i.e. an update in our context) and a set of hashes, which resolve to its predecessors. (like Git)
- The ancestor relation between the operations in the hash DAG reflects the causal dependency between the operations.



Theorem

The heads of a hash DAG is the set of nodes with no successors. If two peers have the same set of heads, their hash graphs, including CRDT operations contained in the hash graphs must be same.

Convergence

Exploiting validity check

- Traditionally, a CRDT operation's validity is checked before it is broadcast to the peers. However, a malicious peer can skip the validity check to cause problems.

Hllo

p

⟨Insert "e" after "X" ⟩ invalid!

Convergence

Exploiting validity check

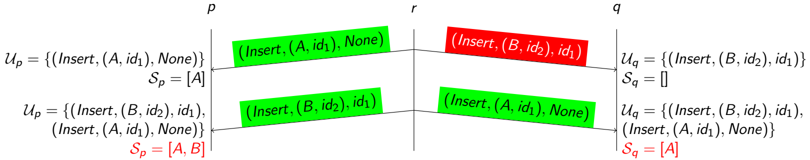
- Traditionally, a CRDT operation's validity is checked before it is broadcast to the peers. However, a malicious peer can skip the validity check to cause problems.
- Instead of relying on validity at the sender side, we shift the validity check responsibility to receiver side.

Convergence

RGA as an example

However, naively moving the validity check to the receiver side could cause inconsistent states.

$(Insert, (A, id_1), id_2)$ means insert A with ID id_1 after the element with ID id_2. If id_2 is $None$, then A is inserted as the first element.



Convergence

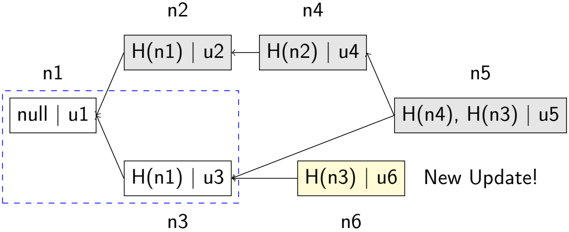
Use ancestor-only checks

- If we can ensure that all peers make the same validity decision, then we can avoid such problems.

Convergence

Use ancestor-only checks

- If we can ensure that all peers make the same validity decision, then we can avoid such problems.
- A way to achieve this is to have each peer only validate operations based on the ancestors of the operation in the hash DAG. This makes the validity check deterministic since all peers share the same ancestors of the operation.



Convergence

Use ancestor-only checks

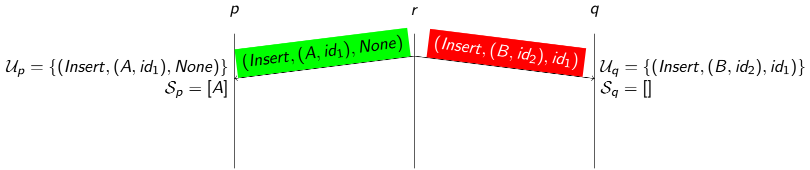
Theorem

If all correct peers follow the protocol, and each operation's validity is checked only against its ancestors in the hash DAG, then a valid operation on one correct peer must also be a valid operation on all other correct peers.

Convergence

Use ancestor-only checks

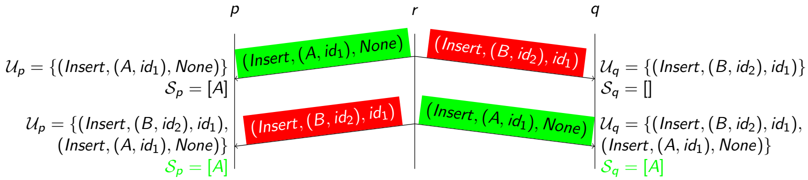
Even though p has element with ID id_1, it still reject the update because the element with ID id_1 is not an ancestor of $(Insert, (B, id_2), id_1)$.



Convergence

Use ancestor-only checks

Even though p has element with ID id_1, it still reject the update because the element with ID id_1 is not an ancestor of $(Insert, (B, id_2), id_1)$.



Convergence

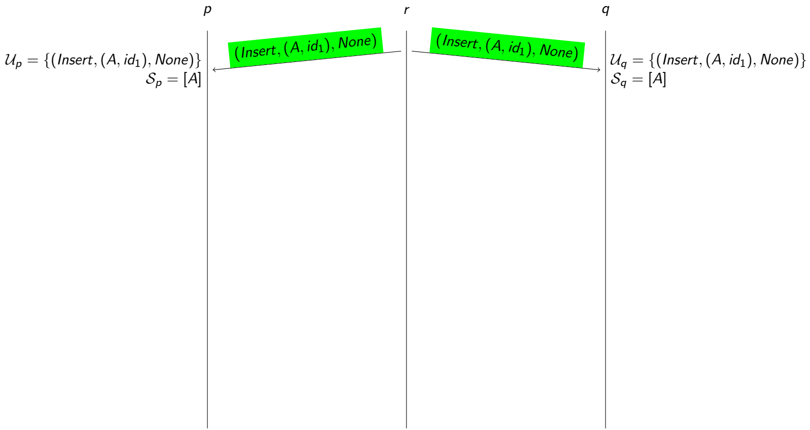
Exploiting uniqueness of IDs

Many CRDT algorithms require unique IDs[2]. It is often achieved by (unique peer-id, counter) traditionally, but a malicious peer can simply use the same (unique peer-id, counter) for different operations, resulting in divergence.

[2] Including ORSet, RGA, Logoot, Treedoc, etc.

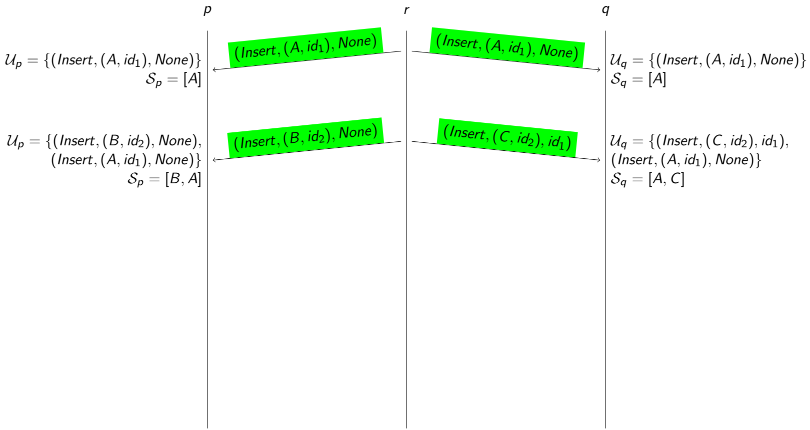
Convergence

RGA as an example



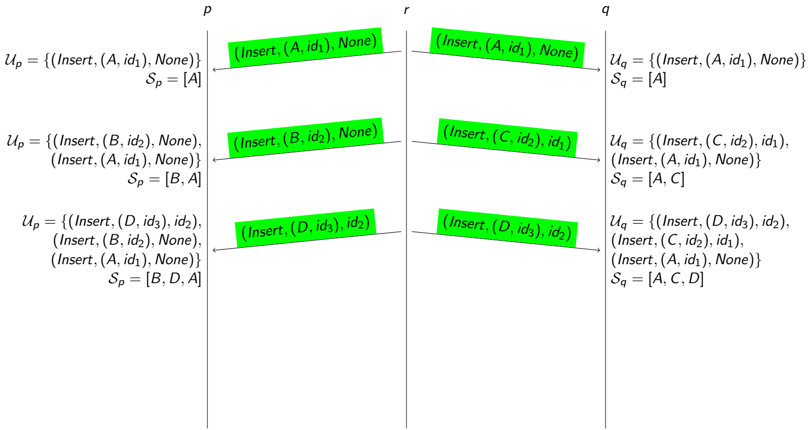
Convergence

RGA as an example



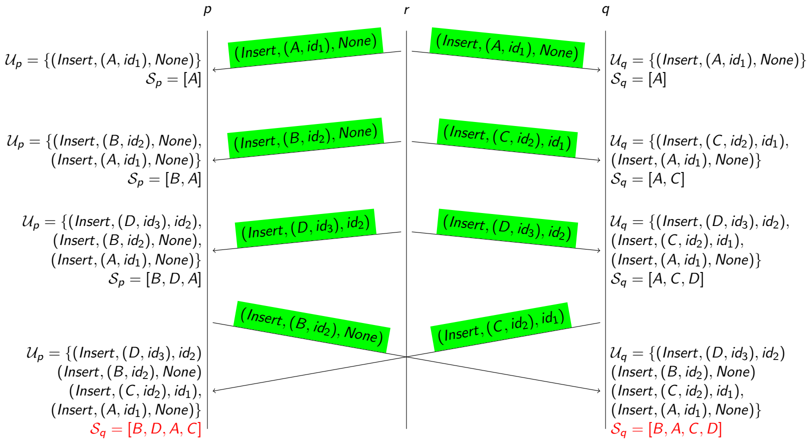
Convergence

RGA as an example



Convergence

RGA as an example



Convergence
Use hash as IDs

- Instead of relying on the sender to provide an unique ID, we let each peer generate the ID by using the hash of the node that contains the operation.
- Since the hash function is collision-resistant, and the nodes in the hash DAG are unique, the IDs are unique.

Questions

- How to prove the theorems we stated in the previous sections in Byzantine envionment?
- How to ensure we have ruled out all possible vulnerabilities for a particular CRDT?

Our Contributions

- We model a peer-to-peer system consisting of indefinitely many peers in Isabelle/HOL, a formal proof assistant.

[3] L Da and M Kleppmann. *A Framework for Designing and Verifying Byzantine Fault Tolerant CRDTs*. Version 0.0.1. https://github.com/LiangrunDa/bft-crdt-isabelle. Oct. 2024.

Our Contributions

- We model a peer-to-peer system consisting of indefinitely many peers in Isabelle/HOL, a formal proof assistant.
- The system only assumes a collision-resistant hash function, and we place no other assumptions on the system.

[3] L Da and M Kleppmann. *A Framework for Designing and Verifying Byzantine Fault Tolerant CRDTs*. Version 0.0.1. `https://github.com/LiangrunDa/bft-crdt-isabelle`. Oct. 2024.

Our Contributions

- We model a peer-to-peer system consisting of indefinitely many peers in Isabelle/HOL, a formal proof assistant.
- The system only assumes a collision-resistant hash function, and we place no other assumptions on the system.
- We proved the theorems that we claimed previously in Isabelle/HOL.

[3] L Da and M Kleppmann. *A Framework for Designing and Verifying Byzantine Fault Tolerant CRDTs*. Version 0.0.1. `https://github.com/LiangrunDa/bft-crdt-isabelle`. Oct. 2024.

Our Contributions

- We model a peer-to-peer system consisting of indefinitely many peers in Isabelle/HOL, a formal proof assistant.
- The system only assumes a collision-resistant hash function, and we place no other assumptions on the system.
- We proved the theorems that we claimed previously in Isabelle/HOL.
- Using those theorems, one can verify the correctness of the resulting BFT CRDTs by proving some simple properties.[3]

[3] L Da and M Kleppmann. *A Framework for Designing and Verifying Byzantine Fault Tolerant CRDTs*. Version 0.0.1. `https://github.com/LiangrunDa/bft-crdt-isabelle`. Oct. 2024.

Formalization

To prove a BFT CRDT correct, one needs to prove the following three properties:

- Concurrent operations commute.
- The validity check only depends on the ancestors of the operation.
- It never fails on a valid operation.

Then the following theorems are automatically proved:

theorem *sec-convergence*:
  assumes ‹*heads* (*graph i*) = *heads* (*graph j*)›
  shows ‹*apply-operations* (*delivered-nodes i*) = *apply-operations* (*delivered-nodes j*)›

theorem *sec-progress*: ‹*apply-operations* (*delivered-nodes i*) ≠ *None*›

Evaluation

- We applied our method to two well-known CRDTs, *ORSet* and *RGA*, and obtained two BFT CRDTs.
- We proved the correctness of the two BFT CRDTs formally using only 244 and 522 LoC respectively.

Conclusion

- We analyzed the possible vulnerabilities of traditional CRDTs under Strong Eventual Consistency model.

Conclusion

- We analyzed the possible vulnerabilities of traditional CRDTs under Strong Eventual Consistency model.
- We proposed several approaches to prevent the vulnerabilities.

Conclusion

- We analyzed the possible vulnerabilities of traditional CRDTs under Strong Eventual Consistency model.
- We proposed several approaches to prevent the vulnerabilities.
- We formalized the system in Isabelle/HOL and proved the correctness of the proposed approaches.

Conclusion

- We analyzed the possible vulnerabilities of traditional CRDTs under Strong Eventual Consistency model.
- We proposed several approaches to prevent the vulnerabilities.
- We formalized the system in Isabelle/HOL and proved the correctness of the proposed approaches.
- Our framework along with the theorems can be used to verify the correctness of a BFT CRDT by proving some simple properties. We evaluated our framework on ORSet and RGA.

Backup Slides

locale *bft-strong-eventual-consistency* = *peers-with-arbitrary-history* +
assumes *sem-check-only-ancestors-relevant*:
‹$(\textit{ancestor-nodes-of } n) \subseteq \textit{fset } G \Longrightarrow \textit{is-sem-valid-set}$
$(\textit{ancestor-nodes-of } n)\ n \longleftrightarrow \textit{is-sem-valid } G\ n$›
assumes *concurrent-opers-commute*: ‹*hb.concurrent-ops-commute*
(*delivered-nodes i*)›
assumes *step-never-fails*: ‹$\textit{apply-history}\ ([], \{||\})\ \textit{ns} = (\textit{dn}, G) \Longrightarrow$
$\textit{no-failure dn} \Longrightarrow$
$\textit{check-and-apply}\ (\textit{dn}, G)\ (\textit{hs}, v) = (\textit{dn}', G') \Longrightarrow \textit{no-failure dn}'$›