RChain Consensus

Michael Birch

April 2018

Outline

1. General CBC Casper Protocols
   - The building blocks
   - Safety
2. Casper for RChain
   - Consensus Values and Protocol States
   - Proof-of-Stake Details
3. Remaining Questions

What is Correct-By-Construction Casper?

- A general framework for defining asynchronously safe, byzantine fault tolerant consensus protocols
- I.e. CBC Casper is not a single protocol, but a class of protocols
- All protocols in the Capser family share the same basic structure and safety proof

What Does a CBC Casper Protocol Need?

- A set of possible consensus values, C
- A logic for making statements about the consensus values
- Protocol states and executions which together define a category
- A function, $\mathcal{E}$, mapping protocol states to true propositions about the current consensus value

A Note About Non-Triviality

- Must have $|C| > 1$
- From the initial protocol state it must be able to select any element of C
- I.e. the protocol can't be "always pick $c \in C$" for some particular c
- This will be important once we see the safety proof

The Typical CBC Protocol

- A protocol state is equal to a set of messages that have been received having fewer than some amount of faults
- Protocol executions are sending/receiving new messages
- Each message includes a sender, an "estimate", and a justification
- A justification is a set of messages that sender has seen
- The protocol demands estimate $= \mathcal{E}$(justification)

Equivocations

- Since all messages have justifications, they can be causally ordered (even in a full asynchronous setting!)
- A single actor must have a serial order to their messages
- Otherwise, they are "equivocating"
- Equivocations are detectable via justifications as a pair of messages from the same sender that cannot be causally ordered with respect to one another
- An equivocation is a byzantine fault

Estimate Safety

- An estimate p is "safe" in a protocol state σ if for all future states σ', $\mathcal{E}(\sigma') \vdash p$
- Note σ is a future state of itself by the identity protocol execution
- This is a *local* property – it only talks about the view of a single actor
- In the typical formulation this means that no other (non-faulty) message I could receive in the future could convince me that p does not hold for the consensus value

Consensus Safety

- An estimate is "consensus safe" if it is consistent with estimates of all future states of all protocol-following actors
- This is a *global* property – it applies to the entire network
- An estimate can be considered "finalized" when it is consensus safe

The Consensus Safety Theorem

Theorem

Estimate Safety implies consensus safety over all protocol states which share a common future.

The Consensus Safety Theorem

In symbols:

$$\sigma_1 \sim \sigma_2 \implies \neg(S(p, \sigma_1) \wedge S(\neg p, \sigma_2))$$

How do we know when we have estimate safety?

- "Safety oracles" are algorithms which can determine if an estimate is safe
- Typically they are not able to determine if an estimate is not safe though
- I.e. $SO(p) = T$ means p is definitely safe; $SO(p) = F$ means we don't know if it's safe or not yet
- Such algorithms already exist, such as the "clique safety oracle"

A Note About Non-Triviality (Revisited)

- If protocol states were just sets of messages, all states would have a common future
- Then estimate safety would always imply consensus safety
- So either there would be no safe estimates or the protocol would be trivial (i.e. there would be no conflicting estimates)
- This is why it is important that protocol states cannot accept more than some number of byzantine faults, it makes some states not have common futures

What are we coming to consensus on?

- A (partially ordered) sequence of changes made to a Rholang term (the genesis term)
- These changes include adding new code in concurrent composition ("deployments")
- As well as Comm. event reductions

What are the protocol states?

- Following the typical construction – sets of messages with fewer than some number of faults
- Messages consist of blocks containing reference to parent state(s), changes made, new post-state
- Protocol executions are sending/receiving blocks

How are protocol states converted in to estimates?

- GHOST fork choice rule, modified to allow for multiple parents
- GHOST gives the head of the DAG, following it back to genesis gives the sequence of changes made to the Rholang term
- GHOST uses "weights" for each of the different actors in the protocol (called validators)
- Weights come from the information contained in the Rholang term via the "blessed" PoS contract
- Weights change over time through bonding/unbonding; when determining the "score" of a block, the weights in the parent block are used

A brief outline of GHOST

- GHOST has two stages: scoring and traversal
- During scoring, the latest block from each validator (latest defined by justifications; validator set defined by most recent estimate) propagates its creator's weight back through the DAG along parent-child links
- If a block is passed over multiple times then the weights of the different validators are summed
- As an addition step in scoring, a validator's weight is added to a block which includes that validator's latest block as a "step parent"
- During traversal, we start an genesis and move forward through the DAG to the child of the current block which has the highest score until reaching a head of the DAG

Details of Proof-of-Stake

- https://rchain.atlassian.net/wiki/spaces/CORE/pages/346849284/Details+of+Proof-of-Stake+in+RChain

What don't we have worked out yet?

- How many races should be allowed to be decided in a single block? One? N ($N > 1$)? However many it takes to reach quiescence?
- Apart from that, basically everything involving slashing
- Security of the system comes because of slashing
- The goal is to have a system which is "incentive compatible" meaning that the rational decision (in a game-theoretic sense) is to follow the protocol
- Slashing is the most powerful tool we have in shaping the incentives validators have

Attacks and slashing

- Primary attack vector seems to be censorship – validator "pretends" to have not seen some message
- A synchrony constraint could make this an attributable and slashable offense. What should the synchrony assumption be?
- A few other questions related to slashing are given in the "Open Questions" section of the "Details of Proof-of-Stake" wiki page