Project Proposal

Proof-of-Replication Implementation

Dr. Thomas Austin

Professor

Department of Computer Science

Trique (Dung) Nguyen

Software Engineering Undergraduate

CS 168 - Blockchain & Cryptocurrencies

Date of Submission: Apr 24, 2024

1. Background

1.1. Choice of Topic

My area of interest in blockchain consensus is Proof-of-Storage, specifically Proof-of-Replication (PoRep) and Proof-of-Spacetime (PoSt), which Filecoin has been researching and developing. In this class project, I plan to implement the PoRep consensus based on Filecoin's research. If time permits, I will extend the PoRep to PoSt.

1.2. Theory

Proof-of-Replication: a consensus gives proof to the user (verifier V) that some data D has been replicated and stored by the server (prover P) in physical storage.

Proof-of-Spacetime: a consensus ensures the storage provider is storing data at a specific time of challenge.

Filecoin PoRep protocol

Setup

- INPUTS:
  - prover key pair $(\mathsf{pk}_\mathcal{P}, \mathsf{sk}_\mathcal{P})$
  - prover SEAL key $\mathsf{pk}_{\mathsf{SEAL}}$
  - data $\mathcal{D}$
- OUTPUTS: replica $\mathcal{R}$, Merkle root rt of $\mathcal{R}$, proof π_{SEAL}

1) Compute $h_\mathcal{D} := \mathsf{CRH}(\mathcal{D})$
2) Compute $\mathcal{R} := \mathsf{Seal}^\tau(\mathcal{D}, \mathsf{sk}_\mathcal{P})$
3) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
4) Set $\vec{x} := (\mathsf{pk}_\mathcal{P}, h_\mathcal{D}, \mathsf{rt})$
5) Set $\vec{w} := (\mathsf{sk}_\mathcal{P}, \mathcal{D})$
6) Compute $\pi_{\mathsf{SEAL}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{SEAL}}, \vec{x}, \vec{w})$
7) Output $\mathcal{R}$, rt, π_{SEAL}

Prove

- INPUTS:
  - prover *Proof-of-Storage* key $\mathsf{pk}_{\mathsf{POS}}$
  - replica $\mathcal{R}$
  - random challenge c
- OUTPUTS: a proof π_{POS}

1) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
2) Compute path := Merkle path from rt to leaf $\mathcal{R}_c$
3) Set $\vec{x} := (\mathsf{rt}, c)$
4) Set $\vec{w} := (\mathsf{path}, \mathcal{R}_c)$
5) Compute $\pi_{\mathsf{POS}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{POS}}, \vec{x}, \vec{w})$
6) Output π_{POS}

Filecoin PoSt protocol

Setup

- INPUTS:
  - prover key pair $(\mathsf{pk}_\mathcal{P}, \mathsf{sk}_\mathcal{P})$
  - prover POST key pair $\mathsf{pk}_{\mathsf{POST}}$
  - some data $\mathcal{D}$
- OUTPUTS: replica $\mathcal{R}$, Merkle root rt of $\mathcal{R}$, proof π_{SEAL}

1) Compute $\mathcal{R}$, rt, $\pi_{\mathsf{SEAL}} := \mathsf{PoRep.Setup}(\mathsf{pk}_\mathcal{P}, \mathsf{sk}_\mathcal{P}, \mathsf{pk}_{\mathsf{SEAL}}, \mathcal{D})$
2) Output $\mathcal{R}$, rt, π_{SEAL}

Prove

- INPUTS:
  - prover PoSt key $\mathsf{pk}_{\mathsf{POST}}$
  - replica $\mathcal{R}$
  - random challenge c
  - time parameter t
- OUTPUTS: a proof π_{POST}

1) Set $\pi_{\mathsf{POST}} := \perp$
2) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
3) For $i = 0 \ldots t$:
   a) Set $c' := \mathsf{CRH}(\pi_{\mathsf{POST}} || c || i)$
   b) Compute $\pi_{\mathsf{POS}} := \mathsf{PoRep.Prove}(\mathsf{pk}_{\mathsf{POS}}, \mathcal{R}, c')$
   c) Set $\vec{x} := (\mathsf{rt}, c, i)$
   d) Set $\vec{w} := (\pi_{\mathsf{POS}}, \pi_{\mathsf{POST}})$
   e) Compute $\pi_{\mathsf{POST}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{POST}}, \vec{x}, \vec{w})$
4) Output π_{POST}

Verify
- INPUTS:
  - prover public key, $\mathsf{pk}_{\mathcal{P}}$
  - verifier SEAL and POS keys $\mathsf{vk}_{\mathsf{SEAL}}$, $\mathsf{vk}_{\mathsf{POS}}$
  - hash of data $\mathcal{D}$, $h_{\mathcal{D}}$
  - Merkle root of replica $\mathcal{R}$, rt
  - random challenge, c
  - tuple of proofs, $(\pi_{\mathsf{SEAL}}, \pi_{\mathsf{POS}})$
- OUTPUTS: bit b, equals 1 if proofs are valid

1) Set $\vec{x_1} := (\mathsf{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathsf{rt})$
2) Compute $b_1 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{SEAL}}, \vec{x_1}, \pi_{\mathsf{SEAL}})$
3) Set $\vec{x_2} := (\mathsf{rt}, c)$
4) Compute $b_2 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{POS}}, \vec{x_2}, \pi_{\mathsf{POS}})$
5) Output $b_1 \wedge b_2$

Verify
- INPUTS:
  - prover *public key* $\mathsf{pk}_{\mathcal{P}}$
  - verifier SEAL and POST keys $\mathsf{vk}_{\mathsf{SEAL}}$, $\mathsf{vk}_{\mathsf{POST}}$
  - hash of some data $h_{\mathcal{D}}$
  - Merkle root of some replica rt
  - random challenge c
  - time parameter t
  - tuple of proofs $(\pi_{\mathsf{SEAL}}, \pi_{\mathsf{POST}})$
- OUTPUTS: bit b, equals 1 if proofs are valid

1) Set $\vec{x_1} := (\mathsf{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathsf{rt})$
2) Compute $b_1 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{SEAL}}, \vec{x_1}, \pi_{\mathsf{SEAL}})$
3) Set $\vec{x_2} := (\mathsf{rt}, c, t)$
4) Compute $b_2 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{POST}}, \vec{x_2}, \pi_{\mathsf{POST}})$
5) Output $b_1 \wedge b_2$

Figure 1: PoRep & PoSt protocols (Source [1])

2. Implementation

In this class project, I plan to implement the simplified Storage Market, where the Client pays the Storage Miner to store their data and the Storage Miner can prove their action with Proof-of-Replication. During the development, some specific parts be simplified for easier implementation.

2.1. Filecoin Overview

Filecoin protocol consists of Storage Market and Retrievability Market networks, with Client, Storage Miner, and Retrieval Miner.

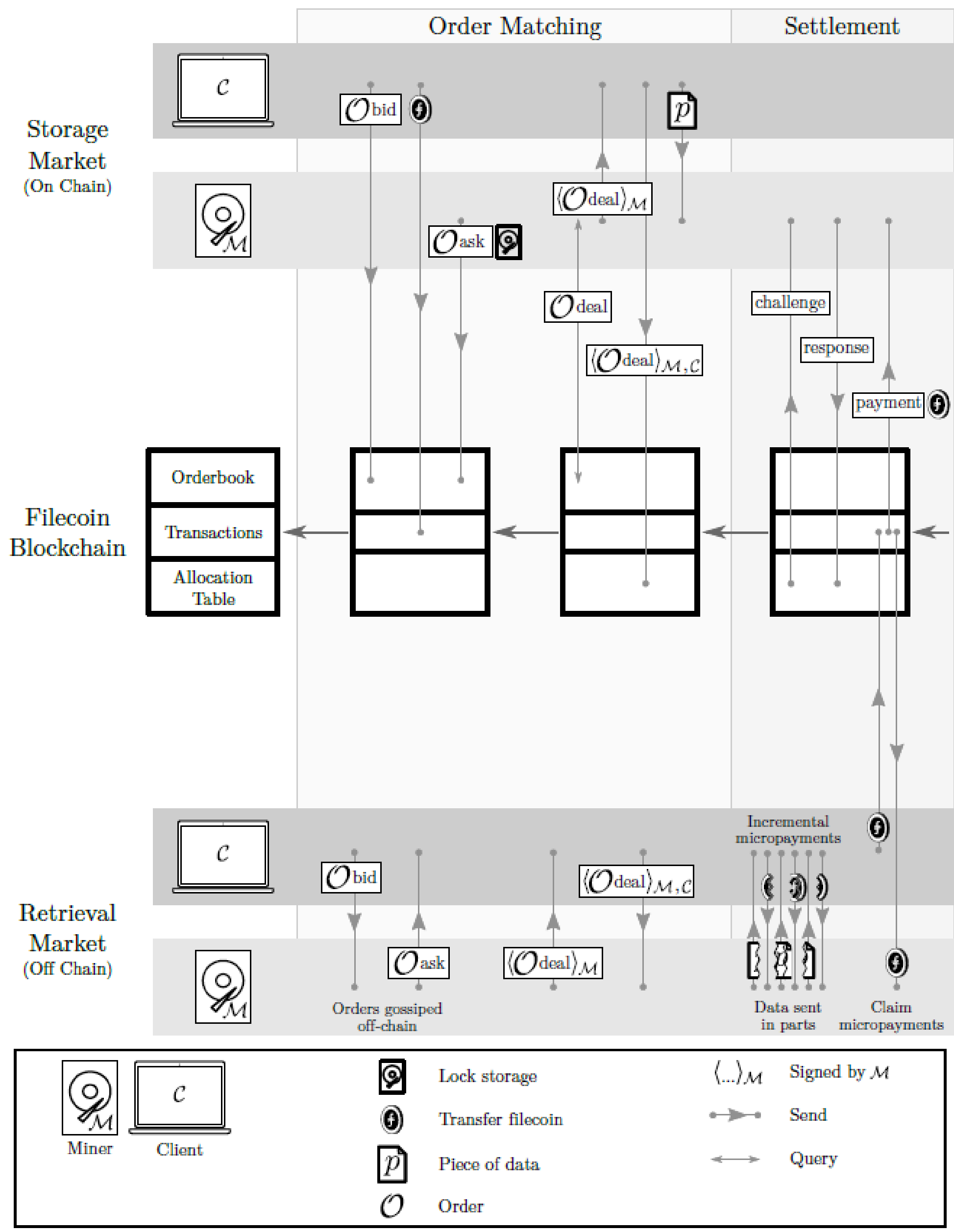

Figure 2: Filecoin Protocol (Source [1])

Put Protocol

Market

AddOrders

- INPUTS: list of orders $\mathcal{O}^1..\mathcal{O}^n$
- OUTPUTS: bit b, equals 1 if successful

1) Set $\mathsf{tx}_{\mathsf{order}} := (\mathcal{O}^1, .., \mathcal{O}^n)$
2) Submit $\mathsf{tx}_{\mathsf{order}}$ to $\mathcal{L}$
3) Wait for $\mathsf{tx}_{\mathsf{order}}$ to be included in $\mathcal{L}$
4) Output 1 on success, 0 otherwise

MatchOrders

- INPUTS:
  - the current Storage Market OrderBook
  - query order to match $\mathcal{O}^q$
- OUTPUTS: matching orders $\mathcal{O}^1..\mathcal{O}^n$

1) Match each $\mathcal{O}^i$ in OrderBook such that:
   a) If $\mathcal{O}^q$ is an *ask* order:
      i) Check if $\mathcal{O}^i$ is *bid* order
      ii) Check $\mathcal{O}^i.\mathsf{price} \geq \mathcal{O}^q.\mathsf{price}$
      iii) Check $\mathcal{O}^i.\mathsf{size} \leq \mathcal{O}^q.\mathsf{space}$
   b) If $\mathcal{O}^q$ is a *bid* order:
      i) Check if $\mathcal{O}^i$ is *ask* order
      ii) Check $\mathcal{O}^i.\mathsf{price} \leq \mathcal{O}^q.\mathsf{price}$
      iii) Check $\mathcal{O}^i.\mathsf{space} \geq \mathcal{O}^q.\mathsf{size}$
2) Output matched orders $\mathcal{O}^1...\mathcal{O}^n$

Exchange

SendPiece

- INPUTS:
  - an *ask* order $\mathcal{O}_{\mathsf{ask}}$
  - a *bid* order $\mathcal{O}_{\mathsf{bid}}$
  - a piece p
- OUTPUTS: a *deal* order $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{M}_i$

1) Get identity of $\mathcal{M}_i$ from $\mathcal{O}_{\mathsf{ask}}$ signature
2) Send $(\mathcal{O}_{\mathsf{ask}}, \mathcal{O}_{\mathsf{bid}}, p)$ to $\mathcal{M}_i$
3) Receive $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{M}_i$
4) Check if $\mathcal{O}_{\mathsf{deal}}$ is valid according to Definition 5.2
5) Output $\mathcal{O}_{\mathsf{deal}}$

ReceivePiece

- INPUTS:
  - signing key for $\mathcal{M}_j$.
  - current orderbook OrderBook
  - *ask* order $\mathcal{O}_{\mathsf{ask}}$
  - *bid* order $\mathcal{O}_{\mathsf{bid}}$
  - piece p
- OUTPUTS: *deal* order $\mathcal{O}_{\mathsf{deal}}$ signed by $\mathcal{C}_i$ and $\mathcal{M}_j$

1) Check if $\mathcal{O}_{\mathsf{bid}}$ is valid:
   a) Check if $\mathcal{O}_{\mathsf{bid}}$ is in OrderBook
   b) Check if $\mathcal{O}_{\mathsf{bid}}$ is not referenced by other active $\mathcal{O}_{\mathsf{deal}}$
   c) Check if $\mathcal{O}_{\mathsf{bid}}.\mathsf{size}$ is equal to $|p|$
   d) Check if $\mathcal{O}$ is signed by $\mathcal{M}_i$
2) Store p locally
3) Set $\mathcal{O}_{\mathsf{deal}} := \langle\, \mathcal{O}_{\mathsf{ask}}, \mathcal{O}_{\mathsf{deal}}, \mathcal{H}(p)\,\rangle_{\mathcal{M}_i}$
4) Get identity of $\mathcal{C}_j$ from $\mathcal{O}_{\mathsf{bid}}$
5) Send $\mathcal{O}_{\mathsf{deal}}$ to $\mathcal{C}_j$
6) Output $\mathcal{O}_{\mathsf{deal}}$

Figure 3: Storage Market Protocol (Source [1])

Manage Protocol

Network

AssignOrders

- INPUTS:
  - deal orders $\mathcal{O}^1_{\mathsf{deal}}..\mathcal{O}^n_{\mathsf{deal}}$
  - allocation table allocTable
- OUTPUTS: updated allocation table allocTable′

1) Copy allocTable in allocTable′
2) For each order $\mathcal{O}^i_{\mathsf{deal}}$:
   a) Check if $\mathcal{O}^i_{\mathsf{deal}}$ is valid according to Definition 5.2
   b) Get $\mathcal{M}_j$ from $\mathcal{O}^i_{\mathsf{deal}}$ signature
   c) Add details from $\mathcal{O}^i_{\mathsf{deal}}$ to allocTable′
3) Output allocTable′

RepairOrders

- INPUTS:
  - current time t
  - current ledger $\mathcal{L}$
  - table of storage allocations allocTable
- OUTPUTS: orders to repair $\mathcal{O}^1_{\mathsf{deal}}..\mathcal{O}^n_{\mathsf{deal}}$, updated allocation table allocTable

1) For each allocEntry in allocTable:
   a) If $t <$ allocEntry.last $+ \Delta_{\mathsf{proof}}$: skip
   b) Update allocEntry.last$= t$
   c) Check if π is in $\mathcal{L}_{t-\Delta_{\mathsf{proof}}:t}$ and PoSt.Verify(π)
   d) On success: update allocEntry.missing$= 0$
   e) On failure:
      i) update allocEntry.missing++
      ii) penalize collateral from $\mathcal{M}_i$'s pledge
   f) If allocEntry.missing $> \Delta_{\mathsf{fault}}$ then set all the orders from the current sector as failed orders
2) Output failed orders $\mathcal{O}^1_{\mathsf{deal}}..\mathcal{O}^n_{\mathsf{deal}}$ and allocTable′.

Miner

PledgeSector

- INPUTS:
  - current allocation table allocTable
  - pledge request pledge
- OUTPUTS: allocTable′

1) Copy allocTable to allocTable′
2) Set $\mathsf{tx}_{\mathsf{pledge}} := (\mathsf{pledge})$
3) Submit $\mathsf{tx}_{\mathsf{pledge}}$ to $\mathcal{L}$
4) Wait for $\mathsf{tx}_{\mathsf{pledge}}$ to be included in $\mathcal{L}$
5) Add new sector of size pledge.size in allocTable′
6) Output allocTable′

SealSector

- INPUTS:
  - miner public/private key pair $\mathcal{M}$
  - sector index j
  - allocation table allocTable
- OUTPUTS: a proof π_{SEAL}, a root hash rt

1) Find all the pieces $p_1..p_n$ in sector $\mathcal{S}_j$ in the allocTable
2) Set $\mathcal{D} := p_1|p_2|..|p_n$
3) Compute $(\mathcal{R}, \mathsf{rt}, \pi_{\mathsf{SEAL}}) := \mathsf{PoSt.Setup}(\mathcal{M}, \mathsf{pk}_{\mathsf{SEAL}}, \mathcal{D})$
4) Output π_{SEAL}, rt

ProveSector

- INPUTS:
  - miner public/private key pair $\mathcal{M}$
  - sector index j
  - challenge c
- OUTPUTS: a proof π_{POS}

1) Find $\mathcal{R}$ for sector j
2) Compute $\pi_{\mathsf{POST}} := \mathsf{PoSt.Prove}(\mathsf{pk}_{\mathsf{POST}}, \mathcal{R}, c, \Delta_{\mathsf{proof}})$
3) Output π_{POST}

Figure 4: Manage Protocol (Source [1])

2.2. SpartanSuperStorage

My initial plan was to implement Proof-of-Replication consensus with SpartanGold but the integration might take a lot of for modifying most of the SpartanGold codebase. I eventually plan to work on the PoRep for my own blockchain implementation based on Filecoin.

For the demo and presentation, my goal is to have a blockchain in which the Client sends some data to the Storage Miner, and the miner stores that data in their pledged storage, which is an associated folder with the miner's information.

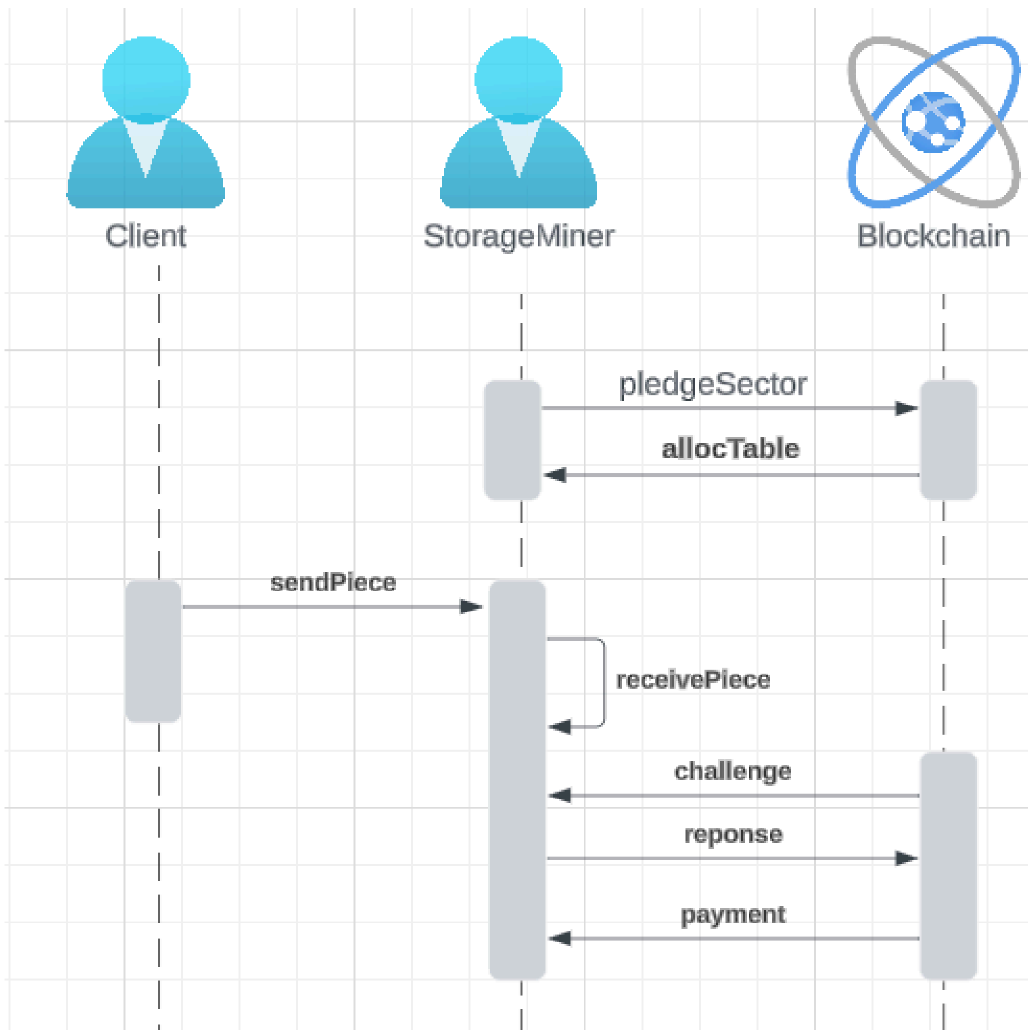

Figure 5: User flow for Demonstration

2.3. Schedule

Date	Task & Plan
Apr 22	Simplified version of PoRep with Seal function
Apr 24	Refined project system, plans & goals Have PoRep functions working
Apr 26	Implement Client & Storage Miner functionalities
Apr 28	Implement Block & Blockchain functionalities
Apr 30	Integrate everything

References

1. Filecoin: A decentralized storage network. Technical report, Protocol Labs, August 2017.