Summary Document on Precision and Determinism in Soda Minting Mechanism Using Arbitrary-Precision Decimal Library

Contact: Moryia

Introduction

This document outlines the strategic implementation of arbitrary-precision decimal arithmetic for the Soda minting mechanism. Utilizing the `shopspring/decimal` library, this approach ensures precise and deterministic financial calculations. The method described below allows Soda to manually control decimal precision in a manner similar to fixed-point arithmetic but with the flexibility of arbitrary precision.

1. Why Ethereum Does Not Use Floating-Point

Floating-point arithmetic is avoided in Ethereum and similarly, Soda does not use it due to:

- **Non-Determinism**: Floating-point calculations can yield different results on various hardware or software setups, unacceptable for blockchain consensus.
- **Precision and Security Risks**: Floating-point can introduce rounding errors and lacks the necessary precision for accurate financial transactions, potentially leading to vulnerabilities.

2. Limitations of `float64` in Minting Code

The use of `float64` is unsuitable for Soda's minting process due to:

- **Inconsistency Across Platforms**: Different platforms may handle `float64` calculations differently, which can compromise blockchain consensus, this might cause external validators to have a different state and thus not be able to validate blocks
- **Precision Loss**: Inability to represent all decimal values exactly, leading to rounding errors in crucial financial calculations. This is not very crucial but nice to have.

3. Inadequacy of Dockerization

While Docker ensures consistent software environments, it does not address the fundamental issues with floating-point arithmetic:

- **Persistent Floating-Point Behaviors**: Docker cannot standardize how floating-point numbers are handled by underlying CPU architectures, which might still lead to variations.
- **Hardware-Specific Behaviors**: Differences in CPU architectures can affect the execution of floating-point operations, even under Docker's standardized environment.

4. Embracing Arbitrary-Precision Decimal Library

Soda adopts the `shopspring/decimal` library to overcome these challenges:

- **High Precision and Flexibility**: The library supports arbitrary-precision which is crucial for maintaining exactitude in financial calculations.
- **Deterministic Calculations Across Platforms**: Ensures consistent results in calculations across any computing environment, bolstering blockchain integrity.
- **Control Over Precision**: Precision is managed operationally rather than set globally, allowing tailored precision management for each specific calculation.

5. Required Changes in Code

To fully integrate the `shopspring/decimal` library in Soda's minting process, several changes are necessary:

- **Initialization from Integers**: Directly convert integer values to `decimal.Decimal` objects, maintaining precision from the start.
  `initialSupplyDecimal := decimal.NewFromInt(initialSupply)`

- **Modify Calculation Logic**: Replace all `float64` operations with `decimal.Decimal` methods, applying precision controls like rounding where needed.

  ```
  mintedTokensDecimal :=
  initialSupplyDecimal.Mul(inflationRateDecimal).Round(2)
  ```

- **Conversion for Blockchain Interaction**: Convert `decimal.Decimal` back to integers or suitable formats for blockchain interactions.
  `amountInWei :=`

```
newBig.Int).SetInt64(mintedTokensDecimal.IntPart())
```

- **Update Data Handling and Logging**: Adapt data handling and logging to reflect operations using high-precision decimal values accurately.

Conclusion

Transitioning to the `shopspring/decimal` library allows Soda to perform financial calculations with high precision and determinism essential for blockchain operations. This shift significantly enhances the security and reliability of the minting process, aligning with the stringent requirements of blockchain deterministic operations.