vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases

Cheng Xu , Ce Zhang , and Jianliang Xu

Department of Computer Science, Hong Kong Baptist University, Hong Kong

{chengxu, cezhang, xujl}@comp.hkbu.edu.hk

Problem Statement

- **Background**: Increasing demand to query blockchain database
- **Blockchain Database Solution**: SAP Leonardo, BigchainDB, SwarmDB, etc.
- **Issue**: Existing solutions rely on a trusted party who can faithfully answer user queries.

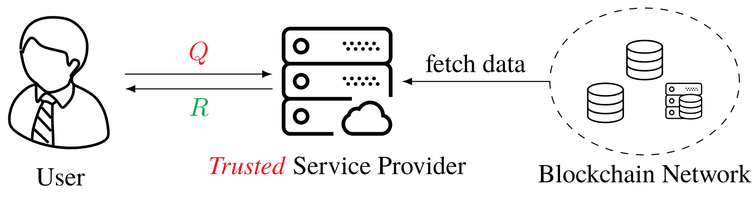

Fig. 1: Workflow of Existing Solutions

- **Question**: How to support integrity-assured queries in untrusted blockchains where a trusted party doesn't exist?
- **Security Requirements**
  - Soundness: none of the objects returned as results have been tampered with and all of them satisfy the query conditions
  - Completeness: no valid result is missing regarding the query conditions
- **Naive Solutions**
  - User becoming full node ⇒ high cost in storage/computation/network
  - Leverage smart contract ⇒ long latency, poor scalability, privacy concern, high cost
- **Our Solution**
  - Miners compute and commit *authenticated data structure* (ADS) in block headers
  - Users become light nodes
  - Queries are outsourced to full nodes
  - Users verify the query results using
    - *Verification Object* (VO) from full nodes
    - ADS from block headers

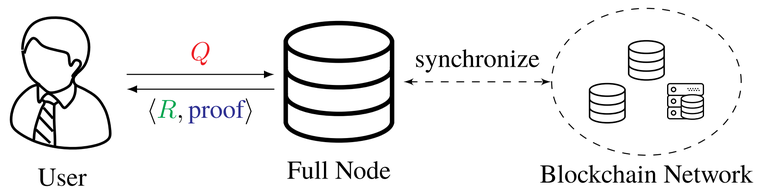

Fig. 2: vChain

Data Model & Queries

- **Data Model**
  - Each block contains several temporal objects $\{o_1, o_2, \ldots, o_n\}$
  - o_i is represented by $\langle t_i, V_i, W_i \rangle$
    (*timestamp, multi-dimensional vector, set valued attribute*)
- **Boolean Range Queries**
  - Find all Bitcoin transactions happening in certain period
    Tx: $\langle$time, transfer amount, {"send address", "receive address"}$\rangle$
    $q = \langle [2018\text{-}05, 2018\text{-}06], [10, +\infty], \text{"send:1FFYc"} \wedge \text{"receive:2DAAf"} \rangle$
  - Subscribe to car rental messages with certain price and keywords
    Tx: $\langle$time, rental price, {"type", "model"}$\rangle$
    $q = \langle -, [200, 250], \text{"Sedan"} \wedge (\text{"Benz"} \vee \text{"BMW"}) \rangle$

Example of Mismatch

- Transform query condition to a list of sets:
  $q =$ "Sedan" $\wedge$ ("Benz" $\vee$ "BMW") $\rightarrow$ {"Sedan"}, {"Benz", "BMW"}
- Consider o_i : {"Van", "Benz"}, we have {"Sedan"} $\cap$ {"Van", "Benz"} $= \varnothing$
- Apply ProveDisjoint({"Van", "Benz"}, {"Sedan"}, pk) to compute proof π
- User retrieves $\mathit{AttDigest} = acc(\{\text{"Van"}, \text{"Benz"}\})$ from the block header and uses VerifyDisjoint($\mathit{AttDigest}, acc(\{\text{"Sedan"}\}), \pi, pk$) to verify the mismatch

Cryptographic Building Block

- **Merkle Hash Tree**
  - Support efficient membership/range queries
  - Limitations
    - An MHT supports only the query keys on which the Merkle tree is built
    - MHTs do not work with set-valued attributes
    - MHTs of different blocks cannot be aggregated

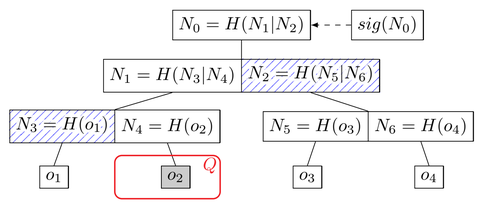

Fig. 3: Merkle Hash Tree

- **Cryptographic Multiset Accumulator**
  - Map a multiset to an element in cyclic multiplicative group in a collision resistant way
  - **Utility**: prove set disjoint
  - Protcols:
    - KeyGen$(1^\lambda) \rightarrow (sk, pk)$: generate keys
    - Setup$(X, pk) \rightarrow$ acc(X): return the accumulative value w.r.t. X
    - ProveDisjoint$(X_1, X_2, pk) \rightarrow \pi$:
      on input two multisets X_1 and X_2, where $X_1 \cap X_2 = \varnothing$, output a proof π
    - VerifyDisjoint(acc(X_1), acc$(X_2), \pi, pk) \rightarrow \{0, 1\}$:
      on input acc(X_1), acc(X_2), and a proof π, output 1 iff $X_1 \cap X_2 = \varnothing$

Extension to Range Queries

- **Idea**: transform numerical attributes into set-valued attributes
- Numerical value can be transformed into a set of binary prefix elements
  - Example: trans$(4) = \{1*, 10*, 100\}$
    * denotes wildcard matching operator

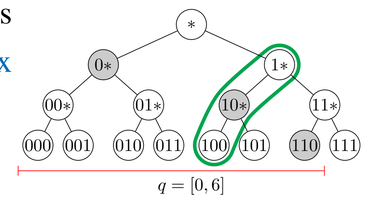

Fig. 5: Example of Transformation

- Range can be transformed into an equivalent boolean expression using a binary tree
  - Example: $[0, 6] \rightarrow 0* \vee 10* \vee 110 \rightarrow$ Equivalence set: $\{0*, 10*, 110\}$
- Range queries can be processed in a similar manner as Boolean queries
  - Transform $v_i \in [\alpha, \beta] \rightarrow$ trans$(v_i) \cap$ EquiSet$([\alpha, \beta]) \neq \varnothing$
  - Example:
    - $4 \in [0, 6] \rightarrow \{1*, 10*, 100\} \cap \{0*, 10*, 110\} = \{10*\} \neq \varnothing$
    - $7 \notin [0, 6] \rightarrow \{1*, 11*, 111\} \cap \{0*, 10*, 110\} = \varnothing$

Batch Verification & Subscription Queries

- **Observation**: objects may share common attributes that mismatch query condition
- **Idea**: we can aggregate them to speed up query processing
  - **Intra-Block Index**: aggregate objects inside same block using MHT
  - **Inter-Block Index**: aggregate objects across blocks using skip list
  - **Inverted Prefix Tree**: aggregate similar subscription queries from users

Basic Solution

- Consider *a single object* and *boolean query*
- Each block stores a single object $o_i = \langle t_i, W_i \rangle$
- **ADS generation** (Miner)
  - Extend the block header with *AttDigest*
  - $\mathit{AttDigest} = acc(W_i) =$ Setup(W_i, pk)
    - Constant size regardless of number of elements in W_i
    - Support ProveDisjoint$(\cdot)$ & VerifyDisjoint$(\cdot)$

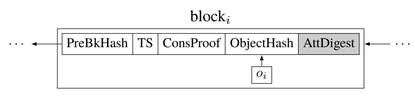

Fig. 4: Extended Block Structure

- **Verifiable Query**
  - **Match**: return o_i as a result; integrity is ensured by the *ObjectHash* in the block header
  - **Mismatch**: use *AttDigest* to prove the mismatch of o_i

Performance Evaluation

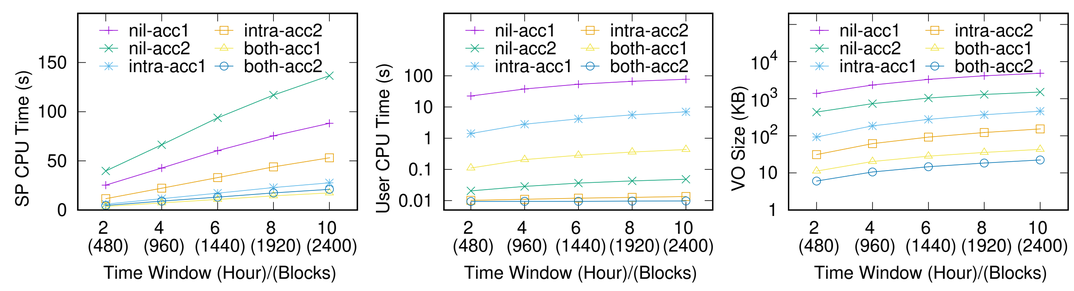

Fig. 6: Time-Window Query Performance over ETH dataset