Hyperledger Indy
Public Blockchain

Hyperledger Bootcamp Russia

Presented by Alexander Shcherbakov
Indy has its own implementation of Distributed Ledger not dependent on any other blockchain platform

Indy has its own implementation of a PBFT-like consensus protocol
Indy is one active Hyperledger projects.
Indy deployment (Sovrin) is in production for more than 2 years.

Sovrin Networks:
- Builder Net
- Staging Net
- Main Net
Agenda

1. Indy-Plenum and Indy-Node
2. Architecture Overview
3. Ledger
4. Consensus Protocol
   - RBFT
   - Moving to Aardvark
   - Plenum protocol specific
5. Summary and Key Features
Indy-Plenum and Indy-Node

- **Indy-Plenum:**
  - [https://github.com/hyperledger/indy-plenum](https://github.com/hyperledger/indy-plenum)
  - Consensus Protocol
  - Ledger

- **Indy-Node:**
  - [https://github.com/hyperledger/indy-node](https://github.com/hyperledger/indy-node)
  - Depends on indy-plenum
  - Identity-specific transactions
Indy-Plenum and Indy-Node

- Indy is a Ledger purpose-build for Identity
- Can be used as a general-purpose Ledger
  - Extend Plenum
  - Custom transactions (pluggable request handlers)
  - Plugins
Indy-Plenum and Indy-Node

- Written in Python
- Depends on
  - ZMQ
  - Indy-crypto (Ursa)
  - Libsodium
- Message-driven and modular architecture
  - Recent refactorings improved this
- Extensive test coverage
  - TDD
  - Unit tests
  - Integration tests
  - Property-based and simulation tests
  - System tests
  - Load tests (usually 25 Nodes)
Architecture Overview: Indy Blockchain Type

- BITCOIN is decentralized money.
- ETHEREUM is decentralized applications.
- INDY is decentralized identity.

<table>
<thead>
<tr>
<th>Access</th>
<th>Permissionless</th>
<th>Permissioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>Bitcoin, Etherium</td>
<td>Indy/Sovrin</td>
</tr>
<tr>
<td>Private</td>
<td>Enterprise Ethereum, Alliance</td>
<td>Hyperledger Fabric, Hyperledger Sawtooth, R3 Corda</td>
</tr>
</tbody>
</table>
Architecture Overview: What data is on Blockchain

- No private data is written to the Blockchain
- Only Public data (such as Issuer’s Public Key) is there
Architecture Overview: Validator and Observer Nodes

- **Validator**
  - Handles Writes and Reads
  - These are the nodes that come to consensus

- **Observer**
  - Handles Reads
  - Keep their “state” in sync with the Validators

*Partially implemented*
Architecture Overview: Validator Nodes

Plenum Consensus Protocol (RBFT)

- Each Node replicates all ledgers
- Each Ledger has a Merkle Tree
- Most of the Ledgers have State based on Patricia Merkle Trie

ZMQ as secure transport
- TCP-based
- CurveCP, libsodium
- Authenticated encryption, no digital signatures
  - Authentication: Poly1305 MAC
  - Symmetric key crypto: XSalza20
  - Public Key crypto: Curve25519

N=3F+1
- N - number of nodes
- F - max number of malicious nodes
Architecture Overview: Write Requests

- (Multi) Signed by the user
- Digital Signature: Ed25519

F+1 equal replies
Architecture Overview: Read Requests

Just 1 Reply:
- BLS multi-sig
- State (audit) proof

Read Request

No signature
Architecture Overview: Authentication

*Authentication is based on the information present in the Ledger*

- **Write Requests:**
  - Must be signed (Ed25519 digital signature)
  - Signature is verified against a Public Key stored on the Ledger (DID txn)
  - Every transaction author must have a DID transaction on the Domain Ledger

- **Read Requests:**
  - Anyone can read, no authentication is required
Architecture Overview: Authorization

*Authorization is based on the information present in the Ledger*

- **Write Requests:**
  - There is a role associated with every DID
  - There are configurable auth rules (stored in Config Ledger) which can define authorization policy for every action
  - The rules may define how many signatures of the given role are required
  - The rules can be composed by OR/AND expressions

- **Read Requests:**
  - Anyone can read, no authorization is required

Add a new SCHEMA:

(1 TRUSTEE) OR (2 STEWARDS)
Ledger: Transaction Log and Merkle Tree

- **Ledger:**
  - Ordered log of transactions
  - Merkle Tree for the whole ledger
  - No real blocks
- **RocksDB as key-value storage**
- **MessagePack for serialization**
- **Ledger catch-up procedure**
  - On Start-up
  - On lagging behind
Ledger: Merkle Tree

1. Merkle Tree Root Hash
   - Ledger Catchup
   - Transaction Validation
2. Consistency Proof
   - Ledger Catchup
3. Inclusion (audit) Proof
   - Reply to written transaction
   - GET_TXN reply
Ledger: Ledger Types

Indy has multiple Ledgers (each with a separate transaction log and a merkle tree):

- **Audit Ledger**
  - Order across ledgers

- **Pool Ledger**
  - Transaction for every Node in the pool
  - Adding, editing, removing nodes

- **Config Ledger**
  - Pool config parameters
  - Used in transaction validation

- **Domain Ledger**
  - Identity-specific transactions
  - Application-specific transactions

- Plugins can add new ledgers
Ledger: Pool Ledger

- A new Pool is built from genesis transactions
- Nodes can be added and removed from the Pool by sending a NODE txn to the Pool Ledger
- Node’s data can be modified by sending a NODE txn to the Pool Ledger
Ledger: Audit Ledger

- **Why**
  - Synchronization between ledgers
    - Global sequence number between ledgers
    - Ledgers are caught up sequentially and one by one
  - Recovering of pool state after startup
  - External audit

- **Audit transaction as a Block:**
  - Batch seq no
  - View no
  - Corresponding ledger root hash
  - Corresponding ledger size
  - Current Primaries

- **Diagram:**
  - Pool ledger
  - Domain ledger
  - Config ledger
  - Audit ledger

1: pool txn
2: domain txn
3: pool txn
4: config txn
5: domain txn
6: domain txn
...
State

- Each Ledger (except Audit Ledger) has a State
  - Pool State
  - Config State
  - Domain State
- Map ordered list of transactions to the current state as dictionary
  - Dynamic Validation
  - Read requests.
- Merkle Patricia Trie (as in Ethereum)
  - Radix Tree + Merkle Tree
  - Ledger Merkle Tree for Lists (orderedtxn log)
  - Patricia Merkle Trie for Dicts
- Key-value storage - RocksDB.
Consensus Protocol: BFT

- No generals trust any other one general
- Each independently decides to attack, if two others also commit to attack
- With four generals, we can have one faulty general, and we can still agree
Consensus Protocol: RBFT

- Byzantine Fault Tolerance
  - Built on RBFT: Redundant Byzantine Fault Tolerance.
  - Improves over PBFT (by Miguel Castro and Barbara Liskov) by executing several protocol instances in parallel
- Better throughput, lower latency than proof-of-work
- Performs better compared to its predecessors under dynamic load and under attack
Consensus Protocol: RBFT Three Phase Commit
Consensus Protocol: RBFT Redundancy with Active Monitoring

Diagram showing a hierarchical structure with clients at the top, connected to nodes labeled Node 0, Node 1, Node 2, and Node 3. Each node contains instances labeled Primary and Replica.
Consensus Protocol: View Change

- Protocol is leader-based
- Leader may behave maliciously
  - Disconnected/Stopped
  - Degraded performance
  - Inconsistent Data (Ledger/State)
- If the Pool realizes that a Leader needs to be changed, it starts a View Change process
  - RBFT has multiple instance of the protocol that compare performance, and decide if master protocol is degraded
- View Change is implemented the same way as in original PBFT paper
  - A variant without digital signatures
- Plenum has a couple of enhancements to make sure the data is consistent during the View Change
Consensus Protocol: View Change

- All transactions that could be potentially ordered on at least one correct Node are eventually ordered on all Nodes.

- View Change procedure:
  - Each node propagates its prepared certificate to other nodes (that is transaction it could potentially ordered).
  - A new Leader decides which transactions need to be re-ordered and do the re-ordering.

Node 1
- Checkpoint: ppSeqNo=100
- Prepared: ppSeqNo=120

Node 2
- Checkpoint: ppSeqNo=100
- Prepared: ppSeqNo=120

Node 3
- Checkpoint: ppSeqNo=100
- Prepared: ppSeqNo=119

Node 4
- Checkpoint: ppSeqNo=100
- Prepared: ppSeqNo=115

Re-order from ppSeqNo=100 till ppSeqNo=120
Consensus Protocol: Moving to Aardvark

- Although RBFT protocol may be quite sensitive to malicious Leaders in some conditions, it’s slower than other PBFT-like protocols
  - $N^3$ vs $N^2$
- We are expecting to change consensus protocol to Aardvark
  - PBFT-like protocol with the same view change implementation
  - Has just 1 protocol instance (like in PBFT and unlike RBFT)
  - Does regular View Changes
  - Probability of View Change depends on the Leader’s performance
Plenum Protocol Specific

- **3PC Batching**
  - Multiple transactions are ordered as one in a batch

- **Data Consistency check as part of Consensus Protocol**
  - Apply batches as proposed by the Leader to the Ledgers and States => uncommitted merkle root
  - Compare uncommitted merkle root hash with the Leader’s one (in PrePrepare message)
  - This guarantees Data Consistency
  - If Leader sends inconsistent Data - View Change happens
Plenum Protocol Specific

- **Dynamic validation based on the current uncommitted state**
  - When a PrePrepare is applied, each transaction must pass the dynamic validation
  - Dynamic validation is performed against the current uncommitted Ledger or State

- **Usage of Audit Ledger**
  - Audit Ledger is used to confirm data consistency as part of consensus
  - Audit Ledger’s root is used as a Checkpoint
Plenum Protocol Specific

- **Sequential applying of PrePrepares**
  - We may have more than one Batch (PrePrepare) in flight, but all PrePrepares are applied sequentially (no gaps) to check data consistency.

- **Message Requests**
  - If a message from a Node is lost/missing, it’s requested from this Node.

```
PrePrepare: ppSeqNo=10
Apply

PrePrepare: ppSeqNo=11
Apply

PrePrepare: ppSeqNo=12
- Apply
  - Apply
    - Apply
      ppSeqNo=12

PrePrepare: ppSeqNo=13
Do not Apply
- Apply
  ppSeqNo=13
```
Plenum Protocol Specific: BLS multi-signature

Sufficient to send Read requests to just one Node:

- State (Audit) Proof
  - Merkle Tree Proof that the result belongs to a State (Ledger) Merkle Tree with the given root
- BLS multi-signature against the merkle tree root
  - All nodes multi-sign the merkle tree root of Ledgers and States as part of Consensus Procedure

The client verifies State (Audit) Proof and BLS multi-sig
Plenum Protocol Specific: BLS multi-signature

- **BLS multi-signature as part of Consensus Protocol**
  - Each Node BLS signs data during Consensus
    - Ledger merkle root hash
    - State merkle root hash
    - Timestamp
  - BLS multi-signature is calculated once the Batch is ordered
  - If there is no requests in the Pool, a PrePrepare with no requests is sent to update the BLS multi-signature

Example of BLS multi-sig calculation for Node 1
The same is applied to every Replica
Cryptography Summary

● Ledgers:
  ○ Merkle Tree (Ledger)
  ○ Patricia Merkle Trie (State)

● Node-to-Node Communication
  ○ ZMQ (libsodium) as secure transport
    ■ CurveCP handshake
    ■ Authenticated Encryption
      ● Authentication: Poly1305 MAC
      ● Symmetric key crypto: XSalsa20
      ● Public Key Crypto: Curve25519
    ■ No Digital Signatures
  ○ BLS multi-signature to sign merkle roots

● Client-to-Node communication
  ○ Ed25519 Digital Signatures
Summary

- Ledger purpose-built for Identity
- Indy has its own Ledger and consensus protocol implementation
- Indy is in production (Sovrin network) for more than 2 years
- Indy Consensus Protocol:
  - RBFT consensus protocol with a plan to move to Aardvark
- Indy Ledger:
  - Multiple Ledgers (each with Merkle Tree)
  - States for efficient reads and validation
  - Authentication, Authorization and dynamic validation is based on the information from the Ledger
  - Audit Ledger synchronizes the ledgers and introduces blocks
Summary

- **Efficient Read**
  - Read data from one Node due to BLS multi-signatures and state proofs

- **Specific of the Protocol:**
  - 3PC Batching
  - Data Consistency check as part of Consensus Protocol
  - Dynamic validation based on the current uncommitted state
  - Usage of Audit Ledger
  - Sequential applying of PrePrepares
  - BLS multi-signature as part of Consensus Protocol
Links

- Plenum and Node:
  - [https://github.com/hyperledger/indy-plenum/blob/master/README.md](https://github.com/hyperledger/indy-plenum/blob/master/README.md)
  - [https://github.com/hyperledger/indy-plenum/tree/master/docs/source](https://github.com/hyperledger/indy-plenum/tree/master/docs/source)
  - [https://github.com/hyperledger/indy-node/blob/master/README.md](https://github.com/hyperledger/indy-node/blob/master/README.md)
  - [https://github.com/hyperledger/indy-node/tree/master/docs/source](https://github.com/hyperledger/indy-node/tree/master/docs/source)

- RBFT:
  - [https://pakupaku.me/plaublin/rbft/5000a297.pdf](https://pakupaku.me/plaublin/rbft/5000a297.pdf)

- Aardvark:
  - [https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/clement/clement.pdf](https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/clement/clement.pdf)

- PBFT: