Smart Contract Architecture - Current Implementation

Overview

The Distli Mesh BC platform includes a complete smart contract execution environment with multi-tenant isolation, offline resilience, and enterprise-grade monitoring.

Smart Contract Virtual Machine

Core Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                     Contract Execution Layer                    │
├─────────────────┬─────────────────┬─────────────────────────────┤
│   Contract VM   │  Gas Metering   │     State Management        │
├─────────────────┼─────────────────┼─────────────────────────────┤
│ • Stack-based   │ • Resource      │ • Persistent storage        │
│   execution     │   tracking      │ • Merkle tree verification  │
│ • Deterministic │ • Gas limits    │ • State transitions         │
│   results       │ • Cost control  │ • Rollback support          │
└─────────────────┴─────────────────┴─────────────────────────────┘
                              │
                              ▼
┌─────────────────────────────────────────────────────────────────┐
│                     Blockchain Integration                      │
├─────────────────┬─────────────────┬─────────────────────────────┤
│  Transaction    │   Events &      │     Multi-tenant            │
│   Processing    │  Messaging      │     Isolation               │
├─────────────────┼─────────────────┼─────────────────────────────┤
│ • Deploy TX     │ • Contract      │ • Network separation        │
│ • Call TX       │   events        │ • State isolation           │
│ • Result        │ • Cross-contract│ • Resource isolation        │
│   storage       │   communication │ • Independent execution     │
└─────────────────┴─────────────────┴─────────────────────────────┘
```

Contract Lifecycle

1. **Deployment**: `ContractDeploy` transaction creates new contract instance
2. **Execution**: `ContractCall` transactions invoke contract functions
3. **State Management**: Persistent state updates with event emission
4. **Result Processing**: Execution results stored in blockchain

Current Contract Types

Trading Contract (`trading` type)

Capabilities:

- Order book management (bids/asks)
- Trade matching engine with price discovery
- Order cancellation and modification
- Real-time market data

Functions:

- `buy(asset, quantity, price)` - Place buy order
- `sell(asset, quantity, price)` - Place sell order
- `cancel(orderId)` - Cancel existing order
- `getOrderBook(asset?)` - Retrieve current order book
- `getTrades(asset?, limit?)` - Get trade history

State Structure:

```
{
  "orderBook": {
    "bids": [{"id": 1, "price": 100, "quantity": 5, "trader": "user1"}],
    "asks": [{"id": 2, "price": 105, "quantity": 3, "trader": "user2"}]
  },
  "trades": [{"price": 102, "quantity": 2, "buyer": "user1", "seller": 
"user2"}],
  "nextOrderId": 3
}
```

Multi-Tenant Contract Execution

Network Isolation

- Each tenant network has independent contract state
- Contract instances are network-scoped
- No cross-tenant contract interaction
- Isolated gas accounting and resource limits

Offline Contract Support

```
Online Mode:      Browser ↔ WebRTC ↔ Peers ↔ Tracker ↔ Enterprise BC
                            ↓
                  Contract execution & state sync

Offline Mode:     Browser ↔ WebRTC ↔ Peers (isolated network)
                            ↓
                  Offline contract execution → localStorage
                            ↓
                  Auto-sync when reconnected
```

State Persistence Architecture

Browser Layer:

- Contract state in localStorage per network
- Offline transaction queue
- State export/import capabilities

Tracker Layer:

- Contract state aggregation
- Cross-network state isolation
- Enterprise BC integration

Enterprise Layer:

- Master contract state storage
- Audit trail and compliance
- Analytics and monitoring

Transaction Types

Contract Deployment

```
Transaction::ContractDeploy {
    id: String,
    contract: SmartContract,
    timestamp: u64,
    sender: String,
}
```

Contract Function Call

```
Transaction::ContractCall {
    id: String,
    call: ContractCall {
        contract_id: String,
        function: String,
        params: serde_json::Value,
        caller: String,
        gas_limit: u64,
    },
    result: Option<ContractResult>,
    timestamp: u64,
    sender: String,
}
```

Execution Results

```
ContractResult {
    success: bool,
    result: serde_json::Value,
    gas_used: u64,
    state_changes: Option<serde_json::Value>,
    events: Vec<ContractEvent>,
    error: Option<String>,
}
```

Gas Metering System

Current Implementation

- Simple gas model: fixed costs per operation
- Gas limits per contract call
- Resource consumption tracking
- Gas accounting in transaction results

Gas Cost Structure

- Contract deployment: Variable based on contract size
- Function calls: Base cost + parameter processing
- State writes: Cost per byte written
- Event emissions: Cost per event

Event System

Contract Events

```
ContractEvent {
    event_type: String,
    data: serde_json::Value,
    timestamp: u64,
}
```

Event Types (Trading Contract)

- `OrderPlaced` - New order added to book
- `Trade` - Order matching executed
- `OrderCancelled` - Order removed from book

API Integration

Browser JavaScript Interface

```
// Deploy contract
blockchain.deploy_contract(contract, sender)

// Call contract function
```

```
blockchain.call_contract(call, sender)

// Query contract state
blockchain.get_contract_state(contract_id)

// Get trading data
blockchain.get_order_book(asset)
blockchain.get_recent_trades(asset, limit)
```

REST API Endpoints

- `GET /api/contracts/{id}/state` - Get contract state
- `POST /api/contracts/{id}/call` - Execute contract function
- `GET /api/contracts` - List deployed contracts
- `GET /api/trading/orderbook` - Get order book data

Current Limitations & Enhancement Opportunities

Architecture Limitations

1. **Single Language Support**: Currently Rust-only VM
2. **Simple Consensus**: Basic PoW instead of BFT
3. **Limited Contract Types**: Only trading contract implemented
4. **Basic Gas Model**: Fixed costs vs. complex metering

Roadmap Enhancements (V4.0)

1. **Multi-Language VM**: WASM, JavaScript, Python support
2. **Advanced Consensus**: Byzantine Fault Tolerant protocol
3. **Enhanced Analytics**: ML-based pattern detection
4. **Developer Tools**: SDKs, IDE plugins, testing frameworks

Security Features

Current Implementation

- Deterministic execution across nodes
- Gas limits prevent infinite loops
- State isolation between tenants
- Transaction signature validation

Network Security

- Contract state encrypted in transit
- Multi-signature support for critical operations
- Time-locked transactions

- Audit trail for all contract interactions

Performance Characteristics

Current Benchmarks

- Contract execution: ~10-100ms per call
- State persistence: ~1-10ms per write
- Event emission: ~1ms per event
- Cross-contract calls: ~5-50ms

Scalability Considerations

- Horizontal scaling via tenant isolation
- State sharding per network
- Parallel contract execution
- Optimistic execution with rollback