Building Applications on the Ethereum Blockchain

Eoin Woods
Endava
@eoinwoodz

licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
Agenda

• Blockchain Recap
• Ethereum
• Application Design
• Development
• (Solidity – Ethereum’s Language)
• Summary
Blockchain Recap
What is Blockchain?

- Enabling technology of Bitcoin, Ethereum, ...
- **Distributed database** without a controlling authority
- **Auditable** database with provable lineage
- A way to **collaborate** with parties **without direct trust**
- **Architectural component** for highly distributed Internet-scale systems
Architectural Characteristics of a Blockchain

- P2P distributed
- Append only “ledger”
- Cryptographic security (integrity & non-repudiation)
- Eventual consistency
- Smart contracts
- Fault tolerant reliability

- (Very) eventual consistency
- Computationally expensive
- Limited query model (key only)
- Lack of privacy (often)
- Low throughput scalability (generally – 10stxn/sec)
What Makes a Good Blockchain Application?

• Multi-organisational
• No trusted intermediary
• Need shared source of state (e.g. transactions, identity)
• Need for immutability (e.g. proof of existence)
• Transaction interactions
• Fairly small data size

• No complex query requirement
• Multiple untrusted writers
• Latency insensitive
• Relatively low throughput
• Need for resiliency

“If your requirements are fulfilled by today’s relational databases, you’d be insane to use a blockchain”
– Gideon Greenspan
What is Blockchain being Used For?

- Verifiable supply chains
- Digital ledger that tracks and protects valuable assets
- Supply chain efficiency
- Derivatives post-trade processing
- Post-trade processing
- Identity management
- Verified data
- Georgia government records
# Public and Permissioned Blockchains

<table>
<thead>
<tr>
<th></th>
<th>Public</th>
<th>Permissioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Latency</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td># Readers</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td># Writers</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Centrally Managed</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Transaction Cost</td>
<td>High</td>
<td>“Free”</td>
</tr>
</tbody>
</table>

Based on: Do you need a Blockchain?  
Karl Wüst, Arthur Gervaisy  
Example Blockchains

- Ethereum
- Bitcoin
- NEO
- MultiChain
- Hyperledger
- Corda
Ethereum
Ethereum

- **Open source** blockchain
- **Founded 2014** after Bitcoin experience – Vitalik Buterin, Gavin Wood, Joseph Lubin
- Swiss **governing foundation**
- “Blockchain **App Platform**”
- Vibrant **ecosystem**
Ethereum Key Concepts

• Blocks
• Consensus
• Smart Contracts
• Events and Logs
• Transactions & Calls
• Ether, Gas, Gas Cost, Fees
Ethereum Overview

Ethereum Full Node

Smart Contracts

RPC Interface

Ethereum Client Application

Ethereum Mining Node

Ethereum Full Node

Ethereum Mining Node

Ethereum Full Node

Ethereum Light Node
Ethereum Blockchain – Txns, State, Receipts

https://blog.ethereum.org/2015/11/15/merkling-in-ethereum
Ethereum Consensus

• Process by which **distributed system decides on state**
• Currently uses a ”**Proof of Work**” system
  • Each (mining) node gathers transactions into candidate block
  • “Ethash” algorithm used to generate a hash at a target difficulty
  • If “first”, broadcast the resulting block
• **Forks can occur** due asynchrony – longer fork used
• **Proof of Stake** approach planned and in trial (Casper)
Smart Contracts

• Stored procedures for Ethereum
• Execute **EVM bytecode**
• Four languages:
  • **Solidity**, LLL, Serpent, Vyper
• EVM code **deploys via a txn**
• **Invoked** from other contracts or off chain **using address**
Events and Logs

- **EVM code is quite isolated** from the outside world
  - no `System.out.println()` ... no `java.io.File`!
- **Events can link** EVM code to the outside world
- Events are types in Solidity
- **Fired from the code** and written to EVM “logs”
- **Clients can observe events** by reading the logs
- Useful for logging and for off-chain communication
Events and Logs

```solidity
event LargeThingHappened(address indexed _who, uint what);
event SmallThingHappened(address indexed _who, uint what);

function causeEvent(uint what) {
    if (what > 100) {
        LargeThingHappened(msg.sender, what);
    } else {
        SmallThingHappened(msg.sender, what);
    }
}
```

Log from Solidity

Web3J makes retrieving events straightforward

```java
Event e = new Event("LargeThingHappened", ...);

// ...
RemoteCall<TransactionReceipt> txn1 = // ...
List<LargeThingHappenedEventResponse> evts =
    getLargeThingHappenedEvents(txn1);
```
Transactions and Calls

**Transaction**
- Mutates state
- Broadcast and mined
- Costs ether ("gas" – see later)
- Asynchronous (returns txn hash)

**Call**
- Read only operation
- Runs “locally”, no broadcast
- No cost
- Synchronous (returns result)

Difference actually in invocation – ethCall vs ethSendTransaction API calls – rather than the contract
Ether, Gas, Gas Cost, Fees

- **Ether** – the cryptocurrency underpinning Ethereum
- **Gas** – the unit used to measure execution of your transaction
- **Gas Cost** – the price of one “gas unit” that you are prepared to pay
  - Set higher gas cost to get faster confirmation
- **Fee** – the (gas * gasCost) cost you pay to run your transaction

You provide Ether with your transaction invocation. Gas Cost amount is deducted and sent to the miners, balance is refunded to you.

2018/03/25 – est. gas cost is 2 Gwei (0.000000002 Ether ~= 0.0001c)
Practical Costs on MainNet in March 2018

• Gas is charged for instructions executed and storage used

• Executing a contract costs 21,000 gas + cost of op codes used
  • Example: creating Greeter and Mortal costs 279,165 gas to create
  • Gas price of 2 Gwei => cost of 0.0006 ETH (~= $0.30 USD)

• Storage costs 20,000 gas per 256bit word (625,000,000 per MB)
  • Gas price of 2 Gwei => $1.25 per MB (AWS costs $0.10/GB/month)
  • Block gas limit of ~8,000,000 => 400 words/block (~12KB)

• Ethereum is orders of magnitude more expensive than cloud

• Test or private networks can obviously sidestep this problem

https://hackernoon.com/ether-purchase-power-df40a38c5a2f
Rimba et. al. “Comparing Blockchain and Cloud Services for Business Process Execution” – ICSA 2017
ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

BYZANTIUM VERSION fadb37b - 2018-03-20

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. Introduction

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated through the power of the default, consensus mechanisms, and voluntary respect of the social contract, that it is possible to use the internet to make a decentralised value-transfer system that can be shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographi-

is often lacking, and plain old prejudices are difficult to shake.

Overall, we wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

1.2. Previous Work. Buterin [2013a] first proposed the kernel of this work in late November, 2013. Though now embedded in many more, the key functionality of a block
Application Design
Ethereum Application

```
contract Greeter is Mortal {
    /* define variable greeting of the type string */
    string greeting;
    /* ctor to set message */
    function Greeter(string _greeting) public {
        greeting = _greeting;
    }
    /* function to return greeting */
    function greet() public constant returns (string) {
        return greeting;
    }
}
```
Java & Ethereum Application

API / User Interface

POJO Transaction Processing

Web3J

Database Driver

DApp

contract Greeter is Mortal {
    /* define variable greeting of the type string */
    string greeting;
    /* ctor to set message */
    function Greeter(string _greeting) public {
        greeting = _greeting;
    }
    /* function to return greeting */
    function greet() public constant returns (string) {
        return greeting;
    }
}
Some Key Decisions

• Type of **blockchain deployment** (public, private, public permissioned)
• Smart contract **development environment**
• **Where** is each type of data?
  • On the blockchain? On distributed storage? In a database?
  • Who trusts which piece?
• **What do the smart contracts do?** What does Java do?
• How do smart contracts **interact with the outside world?**
• **Identity** and key management
Communicating with the World - Oracles

**Smart Contract**

```
event Transfer(address indexed _from, address indexed_to, uint256 _value);

function MetaCoin() {
    balances[tx.origin] = 10000;
}

function sendCoin(address receiver, uint amount) returns(bool sufficient) {
    if (balances[msg.sender] < amount) return false;
    balances[msg.sender] -= amount;
    balances[receiver] += amount;
    Transfer(msg.sender, receiver, amount);
    return true;
}
```

**Oracle Processor**

- `sendTransaction(...)`
- `write`
- `getE1Events(...)`

**Data Sources**
Identity

• Need to **manage identity** of requests to the blockchain
• Ethereum identity is EC **public/private keypairs**
• **Similar** problem to **authenticating via middleware** in enterprise apps
  • Does the application “impersonate” the end-user?
  • Does the end user give up their credentials to the application?
  • Does the application use an identity server?
• Application can rely on **Ethereum node** to perform txn signing
• Application can **sign locally** (requiring private keys in a wallet)
• Needs careful consideration quite early
Development
Java and Solidity Development Example

Java source and tests

Solidity source & Javascript tests

Java stubs

JAR files

Linux deployment

.Blockchain

.bin & .abi files
## Types of Blockchain for Development

<table>
<thead>
<tr>
<th>Type</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulators</td>
<td>Ganache, Embark</td>
</tr>
<tr>
<td>Lightweight Nodes</td>
<td>Ethereumjs-vm, Pyethereum</td>
</tr>
<tr>
<td>Local Regular Blockchains</td>
<td>Geth, Parity</td>
</tr>
<tr>
<td>Hosted Nodes or Chains</td>
<td>Infura, Azure</td>
</tr>
<tr>
<td>Public Testing Blockchains</td>
<td>Rinkeby, Ropsten</td>
</tr>
<tr>
<td>Public Blockchain</td>
<td>Mainnet</td>
</tr>
</tbody>
</table>
Example Dev Environment

Java source & tests

Solidity source

Contract Unit Tests

Build Tool (e.g. Gradle)

Unit Test

Int. Test

Production

Ganache (local net)

Geth (testnet)

Geth (mainnet or prod)
Demonstration
Solidity – Ethereum’s Language
pragma solidity ^0.4.9;

contract Mortal {
    address owner;
    function Mortal() public { owner = msg.sender; }
    function kill() public { if (msg.sender == owner) selfdestruct(owner); }
}

contract Greeter is Mortal {
    string greeting;
    function Greeter(string _greeting) public { greeting = _greeting; }
    function greet() public constant returns (string) {
        return greeting;
    }
}
pragma solidity ^0.4.19;

contract SimpleToken {
    int64 constant TOTAL_UNITS = 100000;
    int64 outstanding_tokens;
    address owner;
    mapping (address => int64) holdings;

    function SimpleToken() public {
        outstanding_tokens = TOTAL_UNITS;
        owner = msg.sender;
    }

    event TokenAllocation(address holder, int64 number, int64 remaining);
    event TokenMovement(address from, address to, int64 v);
    event InvalidTokenUsage(string reason);
Simple Token Contract – Allocate Tokens

```solidity
function getOwner() public constant returns(address) {
    return owner;
}

function allocate(address newHolder, int64 value) public {
    if (msg.sender != owner) {
        InvalidTokenUsage('Only owner can allocate tokens');
        return;
    }
    if (value < 0) {
        InvalidTokenUsage('Cannot allocate negative value');
        return;
    }
    if (value <= outstanding_tokens) {
        holdings[newHolder] += value;
        outstanding_tokens -= value;
        TokenAllocation(newHolder, value, outstanding_tokens);
    } else {
        InvalidTokenUsage('value to allocate larger than outstanding tokens');
    }
}
```
Simple Token Contract – Move Tokens

```solidity
function move(address destination, int64 value) public {
    address source = msg.sender;
    if (value <= 0) {
        InvalidTokenUsage('Must move value greater than zero');
        return;
    }
    if (holdings[source] >= value) {
        holdings[destination] += value;
        holdings[source] -= value;
        TokenMovement(source, destination, value);
    } else {
        InvalidTokenUsage('value to move larger than holdings');
    }
}
```
Simple Token Contract – Getters & Fallback

```solidity
def myBalance() constant public returns(int64) {
    return holdings[msg.sender] ;
}

def holderBalance(address holder) constant public returns(int64) {
    if (msg.sender != owner) return ;
    return holdings[holder] ;
}

def outstandingValue() constant public returns(int64) {
    if (msg.sender != owner) return ;
    return outstanding_tokens ;
}

def() public {
    revert();
}
```
Aside: EVM Memory (Storage, Memory, Stack)

<table>
<thead>
<tr>
<th>Storage</th>
<th>Permanent contract state storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Storage on the blockchain</td>
</tr>
<tr>
<td></td>
<td>Expensive (20k gas/word to set, 5k gas/word to update, 200 gas/word to read)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th>Temporary storage during contract execution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not store on blockchain</td>
</tr>
<tr>
<td></td>
<td>Cheap (3 gas to read/write per word)</td>
</tr>
<tr>
<td></td>
<td>Used for ”structs”, arrays and mappings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack</th>
<th>Used for value types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Similar cost to ”memory”</td>
</tr>
</tbody>
</table>
Summary
Building Applications on Ethereum

- **Blockchain** can provide **highly distributed, p2p, resilient** data store and code execution environment – with significant tradeoffs
- Ethereum is a maturing **public and private** blockchain platform
- Ethereum “**dapps” can be integrated into “real” Java applications**
- Solidity development is still **maturing** but tools exist
  - Truffle, Embark, Web3J, Metamask, ...
- Decide answers to **key design questions** early
  - Environment, data storage, Solidity development pipeline, Java vs Solidity, external interaction, identity and key management
- Possible to build applications we have **never built before**
Further Information (1)

• Fundamentals
  • https://ethereum.github.io/yellowpaper/paper.pdf

• Network Tools
  • https://ethstats.net
  • https://ethgasstation.info
  • https://etherconverter.online
  • https://etherscan.io
Further Information (2)

• Developing Solidity Contracts
  • https://github.com/ConsenSys/smart-contract-best-practices
  • http://solidity.readthedocs.io

• Dev Tools
  • Solidity Editors list - https://solidity.readthedocs.io/en/develop
  • Metamask - https://metamask.io
  • Truffle Framework - http://truffleframework.com
  • Embark Framework - https://github.com/embark-framework/embark
  • Web3J - https://web3j.io
  • Web3.js - https://github.com/ethereum/web3.js
Further Information (3)

• Security Tools
  • Patterns - https://github.com/OpenZeppelin/zeppelin-solidity
  • Lint style tool - https://github.com/duaraghav8/Solium

• External Oracles
  • http://www.oraclize.it/

• Data Storage
  • https://ipfs.io/
  • https://swarm-guide.readthedocs.io
Further Information (4)

• Other Interesting Links
  • Ethereum reading list - https://github.com/Scanate/EthList
  • Solidity reading - https://github.com/bkrem/awesome-solidity
  • LLL introduction - https://media.consensys.net/an-introduction-to-lll-for-ethereum-smart-contract-development-e26e38ea6c23
  • Vyper site - https://github.com/ethereum/vyper
  • Blockchain comparison - https://www.nctatechnicalpapers.com/Paper/2017/2017-comparing-blockchain-implementations
Acknowledgements

• Cloud Computing icon by Evgeniy Kozachenko from the Noun Project https://thenounproject.com/term/cloud-computing/204953

• Blockchain icon on slides 24, 25, 32 from https://www.smartcontractthailand.com

• Blockchain icon on slide 31 from https://www.draglet.com/blockchain-applications/private-or-public-blockchain

• Other icons licensed from IconFinder under free to use and commercial licenses

• Logo icons extracted from public websites of the logo owners
Thank You

Eoin Woods
Endava
eoin.woods@endava.com
@eoinwoodz