Blockchain Notes

Josh Snider

April 16, 2018

1 Introduction

These are my notes from reading the *Blockchain Applications A Hands-On Approach* book by Arshdeep Bahga and Vijay Madisetti. A major goal of reading this book, is to write a blog post based on it, either as a review of the book or as a summary of topics it discusses.

1.1 Book Contents

The book starts with an overview of the blockchain and some application templates that can be done with it. It then goes to talk about Ethereum, including how to set up a development environment for it. It then goes to talk about services that can be built on the blockchain such as smart contracts, decentralized applications, mining, whisper, and swarm, before wrapping up with an overview of some more advanced topics involving the blockchain. The book requires minimal understanding of the blockchain on the part of the reader, but assumes a competency with programming. There's a companion website at blockchain-book.com and the source code can be found at `http://www.hands-on-books-series.com/assets/Bahga-Madisetti-Blockchain-Book-Code.zip`.

One thing I would have liked to change about the book is to add "homework" to it. It's pretty close to being able to pass as a college textbook, if it had gone deeper into certain subjects and then had review questions and project ideas at the back of the chapters I could definitely see this being the assigned book for a college class.

1.2 About the Authors

Both of the authors are academics, Vijay's a professor and Arshdeep's a researcher. This book is actually part of a *Hands-On Approach* series which has already covered cloud computing and big data.

2 Blockchain Basics

The main idea behind the blockchain is to create a platform for handling transactions that doesn't depend on a central hub like banks. The first cryptocurrency using a blockchain was Bitcoin which was introduced in 2008 by Satoshi Nakamoto. Without a central authority, there needs to be a new way to prevent people from spending the same money more than once, the Bitcoin solution requires two things, every transaction being announced publically and a system for agreeing on the order of transactions. This ledger works as a chain of blocks or "blockchain" if you will. The process by which blocks are added is called "mining", it involves a node in the network collected the new transactions and then stamping its block with a proof-of-work. In order to incentivize "mining", blockchains routinely award money to the node who mines the block that gets added to the chain.

In 2013, Vitalik Buterin proposed a new type of blockchain network called Ethereum. The main idea is to have a single-programmable blockchain that could be shared by multiple applications as opposed to different cryptocurrencies having different blockchains. Each application takes the form of a "smart contract".

Gavin Wood was the first to create a functional implementation of Ethereum in 2014 which included the programming language Solidity.

Each block has four fields, a timestamp, the hash of the previous block, the hash of the block itself (called a *nonce* value), and the hash of the root of the Merkle tree containing the transactions. This means the the block technically does not contain the transactions it has.

So, what are the core features of the block chain?

1. **Immutability** - Once a transaction is recorded in a block, it can't be deleted or altered unless a majority of nodes are conspiring to do so.
2. **No Central Authority** - This reduces the need for transaction fees and speeds up settlement times. Also, makes it impossible to regulate what people spend money on.
3. **Secure and Transparent** - Transactions are final unless a majority of miners collude to change them. The entire process by which blockchains run are largely specified in publically accessible white papers and implemented by open-source programs run by the public.
4. **Privacy** - While all of the transactions are recorded publically, the private key for each party is kept secret and there's no link between the public key and the identity of the transactors.
5. **Scalable and Available** - The blockchain is replicated amongst nodes to ensure, that even as they leave and join the network the blockchain is preserved.

3 Example Smart Contract

Let's consider an escrow contract between two people. The buyer deposits money in an escrow account, the seller then gives the buyer the item, and the escrow account releases the payment to the seller. This can be done in Solidity as a smart contract quite easily. I'm not going to rehash the example completely, but the key points are that contract accounts are different from user accounts, that distributed apps can be used to make nice interfaces to contracts, and that interactions with the contract are validated by miners on the blockchain.

4 A Blockchain Stack

Blockchain is good at maintaining all the transactions in a network, but it's not particularly good at storing large volumes of data or peer-to-peer messaging. That is Ethereum works as a decentralized computing network, but we need something else for decentralized storage, and decentralized messaging. The book recommends Swarm and Whisper respectively.

4.1 Decentralized Computing - Ethereum

Ethereum uses a virtual machine for its execution environment. This environment is called the Ethereum Virtual Machine (EVM), having each node in the network run the same calculation is not efficient, but is necessary to maintain consensus. Ethereum has two forms of accounts, one called Externally Owned Accounts for normal users and contract accounts whose behavior is controlled by associated contract code. You might be worried that letting people compute on the blockchain would bog it down with endless calculations, fortunately there is a solution. Senders are charged a gas fee which increases the more effort their transaction takes. Running on a virtual machine means that the programming language used can be anything as long as it can be compiled to the EVM bytecode.

4.2 Decentralized Storage - Swarm

Decentralized storage and content distribution network, works as a redundant store of Dapp code. Being peer-to-peer means it lacks a single point-of-failure and makes it resistant to DDoS.

4.3 Decentralized Messaging - Whisper

Transient messages between Dapps, which works by subscribing to topics.

5 Blockchain Applications

Everyone loves technology, but only useful technology becomes as popular as blockchain. What are some of the fields that can use blockchain?

- FinTech
- IoT
- Industrial and Manufacturing
- Assets and Inventory
- Energy
- Supply Chain
- Records and Identity
- Healthcare

A lot of the uses talked about in the book seem like hype to me, either where a centralized-approach would work due to the ready available of a central organization, a need for robustness not strong enough to require a cryptographically-secure ledger, or being better suited to a more mundane Internet of Things approach. This is a weakness of the book that my blog post should address, but I'll try to find some examples of blockchain in action where it seems like a good fit. Although if we assume that everyone involved is already an Ethereum user(?), the amount of applications where a smart contract based app is appropriate increases massively.

6 Blockchain Application Templates

Previous chapter introduced the main components of a blockchain stack, a decentralized computing platform (Ethereum), a decentralized messaging platform (Whisper), and a decentralized storage platform (Swarm), with smart contracts and decentralized apps (Dapps) rounding out the stack.

Dapps run on an individual Ethereum node, while Whisper and Swarm work with clients on each node, but communicate outside of the blockchain network.

The design process for a blockchain app is to list the people or concepts involved and analyze their relationships. This is used to create a smart contract, then if it is necessary for usability we create a Dapp using HTML and Javascript.

6.1 Analyzing a Crowdfunding Campaign

What do we want from a crowdfunding campaign app? We want a campaigner to be able to accept donations from anyone and then if the amount of donations meets a target by a deadline, transfer the donations to the campaigner's EOA. This works well as a smart contract, the smart contract is initialized by the campaign organizer with a deadline and goal, backers fund the campaign, and at the end the organizer checks if the funds have been reached, which either triggers a refund or gives the organizer the money. The book has an implementation of this in chapter six.

This is actually a good non-hype example of a blockchain app, mainly because the alternative is Kickstarter which charges an annoying amount of fees.

6.2 Application Templates

- Many-to-one - Like the crowdfunding example, there's a contract owner and every user interacts with them in the same way, for example ticket sellers, crowdfunders, etc.
- Many-to-one - similar to the above but the variables in the smart contract are used to control an IoT device.
- Many-to-many - The standard peer-to-peer system. Can be used for buyers and sellers, uploaders and downloaders, etc.
- One-to-one - Financial transactions, they can be set up to automatically transact with the users' accounts.

7 Setting up Dev Tools

Now we get to the fun part! Setting up a development environment! The book expects you to be using Ubuntu, but that requirement doesn't appear to be mentioned until the installation instructions begin. Despite spending two years at Microsoft, I'm a true believer in the glory of Ubuntu, but more heads-up would have been nice.

I've copied most of the installation code to the install.sh file in this directory, but this chapter is about more than just apt-getting Ethereum. It's also about how to create a private blockchain for testing purposes, a few tools you can use to interact with Ethereum programmatically, some test tools, and a few tools that are useful for creating Dapps.

7.1 Geth

It also recommends storing your data in a /home/ubuntu/ethchain directory, which is a strange recommendation since that's where user accounts go. I think the author is actually using an account called ubuntu, which means you should replace /home/ubuntu with your HOME directory instead. Your address will also be stored as part of the filename in your keystore directory.

If you get an error that says "Failed to unlock developer account: could not decrypt key with given passphrase" you can remove the -dev from the command line. (Currently, I'm not sure if that's the correct way.) If you get an error that says "Failed to write genesis block: genesis has no chain configuration", you need to add a config value to your genesis.json. There's an example of that in the docs at `https://github.com/ethereum/go-ethereum`.

7.2 Python (pyethapp)

The apt-get dependencies for pyethapp were composed entirely of things I already had installed. Given that this book is intended for people with equivalent experience to juniors in college this might be a common situation, assuming you already were using Ubuntu.

Aftere pip installing pyethapp, I got an import error with 'from ethereum import blocks' as per `https://ethereum.stackexchange.com/questions/19104/pyethapp-importerror-cannot-import-name-blocks`. Trying to install pyethapp from the GitHub page got me 'ImportError: cannot import name keccak' from 'devp2p/crypto.py'.

7.3 Solidity

Solidity is a contract-oriented language with features similar to JavaScript. It supports contracts by having many variables relating to the blockchain always available in the global namespace including units for time and amounts of Ether. Most of the information about Solidity in this book can be found in either Appendix A, here is just telling you how to install it as a development tool. Although, geth and pyethapp should have their own Solidity compilers.

7.4 TestRPC

7.5 Mist Ethereum Wallet

Unfortunately unavailable in the Ubuntu Software Center, Mist is a cryptocurrency wallet that can be downloaded from it's GitHub page. The book gives no indication that other Ethereum wallets exist and does not say why you download this one instead of the others, this is definitely a flaw in the book from my point of view.

7.6 MetaMask

7.7 Web3 API

7.8 Truffle

8 Ethereum Accounts

This has been mentioned before both by me and in the book, but Ethereum distinguishes two types of accounts; Externally Owned Accounts (EOAs) which are owned and controlled by the users and Contract Accounts whose behavior is specified by the bytecode that they were created with and which is executed by the block verification process.

8.1 Working with EOAs

Externally Owned Accounts are secured with public/private keypairs where the private key is further secured by being encrypted with a password. Currently, Ethereum uses AES-128-CTR as its encryption algorithm. Creating new EOAs can be done programmatically with geth, but the Mist Ethereum wallet also lets you create new accounts. Most wallets and Ethereum interfaces likely let you do that as well.

The book gives an overview of user account management, which is unnecessary to repeat here.

8.2 Working with Contract Accounts

First thing to do with contracts is to deploy them. This can be done programmatically in Solidity, but can also be done through a built-in tab in Mist Ethereum wallet, which the book recommends.

Once we've deployed a contract we can try to interact with them. This can either be done through the Mist Wallet as shown in the book or programmatically through a *geth* console.

9 Smart Contracts

10 Dapps

11 Mining

12 Whisper

13 Swarm

14 Advanced Topics

This could have been my favorite chapter in the book, but it would need to be fleshed out much more for me to truly love it. It differs from the more application-oriented chapters that take up the middle of the book to discuss more theoretical matters.

14.1 Double-Spending Problem

It's impossible to spend the same cash twice because once you spend it you no longer physically have it. With bank accounts, it's impossible to spend the same money twice because a central authority (your bank) is keeping track and will stop you. With cryptocurrency, where the money takes the form of bits and bytes and there's no central authority, how do we prevent double-spending? We prevent that with two things 1) publically announcing all transactions and 2) having a system to agree on the order of transactions and their sequence. This can be provided by the immutable public ledger where miners work to validate transactions.

14.2 Byzantine Fault Tolerance

The Byzantine Generals' Problem is a classic problem by Lamport and company. The basic idea is that a number of generals are trying to coordinate when to attack a city. With reliable communication and honest generals, this is pretty straightforward. With unreliable communication and possibly-honest generals, it becomes the kind of problem people write PhD theses on.

The authors claim that the proof-of-work system incorporated into the blockchain is a sufficient solution to the Byzantine Generals Problem. The authors don't elaborate on this assertion which I feel is a major flaw in this section.

14.3 Proof-of-work vs. Proof-of-stake

One of the main features of proof-of-work is its difficulty. Mining a Bitcoin block takes time even when using expensive hardware at full throttle. The thing is that difficulty is expensive, both in electricity and hardware. Proof-of-stake is an alternative that is much easier to do. Proof of stake randomly picks a validator to build the next block and the other validators vote either for or against it. Voting with the majority gives you money, voting against loses you money. Ethereum plans to transition to proof-of-stake in the future, but there's no firm timeline for that. Casper is the most likely implementation. There's a FAQ on the Ethereum GitHub about this at `https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ`.

14.4 Consistency, Availability, and Partition Tolerance

If we have a datastore, there are three things we might want it to have. Consistency, where every reader and writer accesses the same thing, availability, where people can always connect to it, and partition tolerance, where the system works even when spread across multiple nodes that are unable to communicate with each other. Unfortunately, there's a theorem about datastores which states that out of consistency, availability,

and partition tolerance, a system can only have two. Traditional SQL chooses consistency and availability which is also called the ACID principles. NoSQL chooses availability and partition tolerance which is also called the BASE principles. The blockchain as a rule chooses availability and partition tolerance, with consistency being achieved eventually as everyone learns about the newest blocks. The possibility of running a NoSQL database on the blockchain that this implies will not be discussed.

14.5 Turing Completeness

Good news! The Ethereum blockchain is Turing-complete, just like every other system that lets you store values, do loops, and do conditionals. This means that it can do any computable function. However, two concerns that need to be addressed are the fact that the people who run the smart contracts aren't the people who made them and that Ethereum needs to actually finish executing some code before it can make the next block. Fortunately, Ethereum uses the same answer for both problems, charging people a 'gas' fee, which will quickly bankrupt anyone trying to experiment with the halting problem on the Ethereum network.

14.6 Greedy Heaviest-Observed Sub-Tree (GHOST)

One of the important differences between the Bitcoin and Ethereum blockchain is how long it takes before a new block is made. For Bitcoin it takes 10 minutes, since that's more secure. For Ethereum, it's around 17 seconds, since that lets people receive confirmations faster. However, it means that it's more likely for multiple people to have produced blocks that have not yet propagated through the system. This has two main downsides. First, all of the effort that goes into stale blocks is wasted and does nothing to protect the integrity of the blockchain. Second, it encourages people to form teams which can become a security risk if they become big enough.

A solution for both of these issues is called Greedy Heaviest-Observed Sub-Tree, where these stale blocks contribute to the score used to determine which chain of blocks is *the* chain of blocks. These stale blocks used to calculate the score are called "uncles". There existance helps solve the first problem and rewarding people for finding them helps solve the second problem.

14.7 Sybil Attack

The essence of a Sybil attack is creating large numbers of fake identities in order to create a disproportiante level of influence in a network. Sybil attacks are not exclusive to smart contracts on the blockchain, they are endemic in all reputation systems, from sock puppets on internet forums, to real-world rumor campaigns, to polls. The book mentions three ways to defeat Sybil attacks. First is to have a list of authorized users (in this case addresses). Second is to require an entry fee to interact with the application. Third is to build a reputation system on top of the blockchain and require a certain level of reputation to interact with the program.

14.8 Mining Pools and Centralization

Without a central authority, blockchains need to rely on a consensus to determine what did and did not happen. Protocols and cryptography can make this consensus resistant to bad actors, but become significantly less effective if a group of miners can control a majority of the network's computing power. Given the way Bitcoin is set up this can be done with a small amount of people willing to invest a disproportionate amount of money into powerful computers such as application-specific integrated circuit (ASICs). One way to discourage this is to use algorithms that are resistant to ASICs.

14.9 Smart Contracts Vulnerabilities

This isn't necessarily an issue with the blockchain or different from normal software vulnerabilities, but given that smart contracts have their own money and behave independently, they're attractive targets for cyber-

criminals. The main weakness cyber-criminals could exploit is "re-entrancy" where a contract allows itself to be called again before it gets a proper response from someone else. This can in fact be really expensive.

14.10 Blockchain Scalability

We would scale the blockchain to compete with Visa and ACH if we could, but can we? Latency in the blockchain is a security feature, where if we reduced it to make things faster it would be easier for centralized groups to dominate. The size of the blockchain is in the tens of GB's, but certain options can be enabled to shrink that. There's been some talk about sharding Ethereum to make it easier to scale, a discussion that can be seen at `https://github.com/ethereum/wiki/wiki/Sharding-FAQ`.

15 Solidity Tutorial

The book has an appendix that contains a Solidity tutorial. It's only five pages, but I think that's an appropriate amount. Solidity is a strongly and statically typed programming language. The basics of programming in it should come naturally to anyone with experience in Java or C. Fortunately, Solidity is open-source so anyone wanting a deep-dive into the particulars of the language can find the documentation at `https://solidity.readthedocs.io/en/v0.4.21/` or the GitHub page at `https://github.com/ethereum/solidity`.