Independent Study Write-Up

Objective:

The objective of my independent study with Mark was to explore the use-cases of blockchains through the Ethereum platform and to develop a simple proof-of-concept smart contract application using the technology afforded by Ethereum. This write-up will provide a cursory, high-level overview of Ethereum and its potential utility, assuming the reader possesses already an understanding of blockchains, followed by an examination of the technical details pertaining to the smart contracts I've written. It is worth noting that the smart contracts I've written are by no means complete and that there are many limitations in their design, which are discussed in another section, but through the course of writing them, I have gained insights into the trade-offs of adopting a blockchain based solution as well as the common problems of the domain itself.

On Ethereum:

Building upon the blockchain technology of Satoshi Nakamoto, Ethereum offers the same transactional capabilities as its predecessor, Bitcoin, in addition to implementing "smart contracts". Whereas Bitcoin could be programmed to contain code within transactions on its blockchain, the utility of doing so was marginal due to the nature of implementation, constraining Bitcoin to be primarily considered as a controversial store of value. Having created and integrated a turing-complete programming language, Ethereum has fully expanded upon the idea of a programmable blockchain. In a sense, Ethereum's platform is analogous to a distributed database capable of running stored procedures to modify its internal state. The key features, aside from programmability, attracting developers and entrepreneurs alike to Ethereum include faster transaction times (decreased block times), the desire to transition from a traditional proof-of-work algorithm powering block creation to proof-of-stake (significantly decreased resource, i.e. electricity, consumption of the network), and greater scalability of the network as a whole.

These features, although still very much works in progress, if fully implemented, may propel greater adoption of the Ethereum platform and its applications to supplant current centralized architectures which we implicitly trust in. Burgeoning use cases are beginning to emerge and time will tell which of these if any, are here to stay.

Sportsbook Betting Smart Contract:

- The latest code can be found here: https://github.com/9-9-0/BetBook

In concept, the smart contracts are to emulate sports book betting in which bets are placed by participants on the binary outcome of a zero sum, two team sport. Using a simplified version of a conventional method of book balancing, a dynamic point spread, that restricts incoming bets. Ideally when bets are closed, the point spread itself will reflect the most probable outcome. The number of bets falling outside this spread to either side will have a 1:1 ratio. Centralized organizers use this method with a vigorish to achieve the greatest profit when the ideal ratio is satisfied. A more thorough explanation can be found in James Surowiecki's *Wisdom of the Crowds*.

Several components are involved to implement this functionality: the bet book smart contract, a mock API returning current states of games, and the smart contract oracle that uses that API. The smart contracts themselves were tested locally using a combination of the Truffle framework, which was used for contract deployment and debugging through their native console, and Ganache, which provided a local testnet for the contracts to be deployed to.

Bet Book Smart Contract

The logic in this smart contract is intended to handle bet placement and payout. The contract is designed with the following states in mind:

- Contract Initialization
  An arbitrary account initializes a contract with a game id, corresponding to a valid id accessible through the API, and a maximum number of bets that can be placed.
- Game Verification
  Immediately upon initialization, the smart contract calls on the oracle to verify that a game with the input id exists and that it is currently waiting to begin.
- Bet Opening
  After the game is validated, the owner of the contract can then open bets.
- Bet Placement*
  Non-owner accounts can then place bets on the home or visiting team for the game linked to the contract. Bets are placed in accordance to the position of the point spread.
- Bet Closing*
  Upon detection of the conclusion of a game through the oracle, the contract will no longer accept bets.
- Outcome Determination*
  Immediately after bets are closed, the contract will enumerate over the placed bets and determine the winners and losers.
- Bet Withdrawal*
  Following outcome determination the winners will be allowed to withdraw their balance from the contract.

* These states were not implemented and the reasons for why can be found in the discussion section.

Mock API and Oracle

To retrieve the state of a game, or any data that lies outside of the blockchain for that matter, a smart contract oracle is used. Oracles interface with APIs external to the blockchain by having non-smart contract code call an API and then having that code push the data into the contract. This is made possible with smart contract events, essentially log entries in blocks. Non-smart contract code is able to watch for the emittance of specific events and respond accordingly. The intuitive procedure is as follows: a smart contract function is called, emitting an event. Non-smart contract code detects this event and calls on the API. After acquiring the data and processing it, the non-smart contract code then calls the appropriate smart contract setter functions, storing it within the contract. Without events, data synchronization with the outside world can also be achieved by simply having the non-smart contract code periodically call the API and push the data into the smart contract.

The mock API uses the NodeJS framework with MongoDB as the database, all hosted on an AWS server. A team, sport, and game are created arbitrarily in sequence. Once a game is registered, searching for it via its id will return the team and sports associated with it, as well as the current state. Additionally, after the game is registered and in waiting, the API can be called to simulate the game, essentially modifying its state to ended and randomly deciding the scores of the home and visiting team.

Discussion

Optimizing the functionality of the smart contracts was not within the scope of this project but was nevertheless kept in the back of my mind. Smart contract execution is not free as execution of code requires the consumption of gas, which is essentially a denomination of Ether. The costs naturally constrain smart contracts to contain simple, highly efficient logic. I had not seriously considered the costs involved with running my code, but this is certainly something that should be revisited.

I ran into issue with the design of the betting contract soon after I had implemented the initialization and oracle. One key issue was the betting system itself after I realized that using a point spread is unnecessary as its primary function is to increase the profit of the bookkeeper. With smart contracts, a bookkeeper is unnecessary as the code functions implicitly as one. In retrospect, the system could have been simplified by adding the bets to a pool, awarding the bettors in a first-come-first-served fashion in the case that for every winning bet, there was not a corresponding losing bet. More analysis is needed so that the system may handle all the complexities involved with bet placement and withdrawal.

Timing issues regarding synchronization of outside game data with the blockchain also ended up being a concern, not to mention the concern of trust in the oracles themselves. For example, I had originally intended for the oracle to update in response to every single bet placed to ensure that the game had not yet started. To accomplish this, additional events and code to handle these events would be needed, increasing the number of function calls and transactions, thus increasing the overall gas cost of placing bets. Another approach was to, as previously mentioned, run the oracle periodically to synchronize data, which then would also require greater gas costs in exchange for timely updates. Would it make sense for a winning bet to return significantly less than expected, the spoils being eaten up by network costs? There are certainly solutions to this, i.e. optimizing smart contract code and design, but many projects have turned towards adopting a hybrid approach to their applications. Computationally intensive models use blockchain primarily for data storage, designating computation to non-blockchain code. This compromises the “code is law” ethos that Ethereum espouses. As with oracles themselves, how can the data being pushed into contracts be fully trusted?

These are issues, generally speaking, facing the Ethereum platform currently. Solutions for scaling the network alongside computation costs and transactions times as well as solutions for creating truly trustless, decentralized applications are being developed. Whether or not it’s possible is beyond me, but from my perspective, I was able to witness the implications of these issues on my own pet project and hopefully in future endeavors I will be better equipped to account for them.