CS237: PROJECT REPORT

DECENTRALIZED CAB SERVICE

BLOCKCHAIN TECHNOLOGY - ETHEREUM PLATFORM

Avinash Nath Aita

aaita@uci.edu

88289773

Kanika Baijal

kbaijal@uci.com

18045193

Abstract and Objectives:

Many have come to believe that an open,trustless blockchain platform like Ethereum is perfectly suited to serve as the shared back end to a decentralized, secure internet - Web 3.0. Our project is an implementation of Cab services using Ethereum based blockchain for transparency. Due to the decentralized infrastructure, the prices and the algorithmic schema is transparent to the parties leading to a better service solution.

In present scenario, the cab service providers face a lot of criticism - for various reasons such as cutting costs and reducing subsidies. Cab companies try to find a balance with fare rising too high, riders using their services reduces and with an oversupply, drivers are not able to reach their targets. In the end neither the riders nor the drivers are happy. We propose a new decentralized approach to deal with these problems (might not be appreciated from Service providers though).

The aim of the project is a model that is able to handle the above mentioned issues. Some of the ideas are as follows:

- When user chooses a pickup and drop location, the user is presented with a fixed price (policy should be decided)
- This request is broadcasted to all the drivers nearby to the pickup location.
- The driver can now propose a new price which he thinks is reasonable enough to be accepted by the rider. (policy should be decided)
- The new updated prices are shown to the rider from all drivers who accepted that ride.
- Different features can be provided to the user at this time including sorting by increasing quotes, or by rating etc (not included in current implementation)

Definitions:

- **Distributed ledger:** A consensus of replicated and shared digital data across different nodes in a peer-to-peer network.
- **Blockchain:** A distributed ledger database that maintains growling list of transactions, events and digital records. These transactions are ordered in blocks which are linked. They are internally represented in condensed form (hash), authenticated and maintained through distributed network of participants using a group consensus protocol. It provides a reliable way of confirming the parties and the transaction involved.
- **Consensus mechanism:** A method to authenticate and validate the transactions / digital records stored on the Blockchain (any distributed ledger) without relying on any central authority.

Literature Survey:

Why Decentralization is important?

Centralized platforms generally have a predictable life cycle. The start phase includes the centralized authority taking major steps to make their services valuable. As platforms move up the adoption S-curve, their power over users and third parties steadily grows. The easiest way to continue growing adopted lies in extracting data from users. Further, users give up privacy and control of their data making them vulnerable to security breaches. The cooperation between platforms become more competencies and further lead to incompatibilities (such as software application incompatibilities). In this project, we consider such scenario of cab services. Today, the cab service providers face a lot of criticism - for various reasons such as cutting costs and reducing subsidies. They try to find a balance with fare rising too high, riders using their services reduces and with an oversupply, drivers are not able to reach their targets. In the end neither the riders nor the drivers are happy. We propose a decentralized approach to deal with these problems- - Ethereum Blockchain Network.

Consensus:

Consensus mechanism / algorithm allows connected nodes in p2p network to work together as a group without any central authority which can survive on member failures. It provides tolerance to failure in Blockchain along with consensual authentication and validation schemes. There are different mechanisms that build this consensus and different algorithms can be chosen in blockchain implementation which defines it (Proof of work, Proof of stake etc.,).

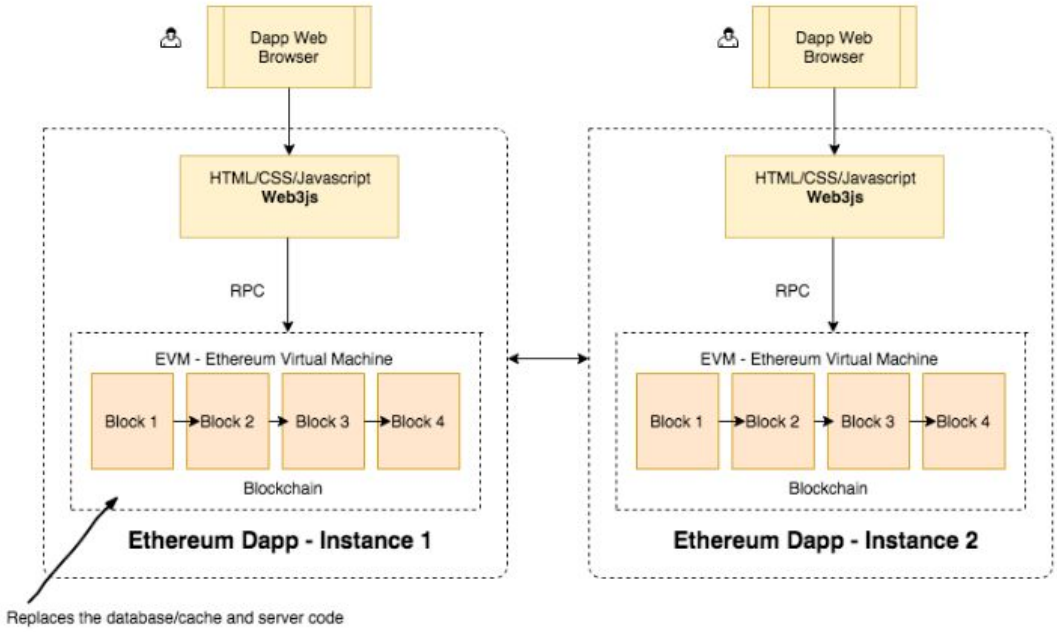
Ethereum:

Ethereum is a public open source Blockchain platform with the ability for users to build programs without the need of middlemen including centralized servers to store information which makes them less exposed to the authorities for abuse. The main intent of Ethereum remains to focus on the concepts on scripting, altcoins and on chain meta-protocols. At the same time it allows developers to create arbitrary consensus-based applications that have the scalability, standardization, feature-completeness and ease of development. Ethereum provides an abstract layer with a built in programming language where anyone is allowed to write smart contracts and decentralized applications . The state in ethereum is made up of accounts each having a 20 byte address and state transitions. ”Ether” is the main internal crypto-fuel of Ethereum, and is used to pay transaction fees.Ethereum maintains mainly two types of accounts which are the Externally owned accounts and the contract accounts. Ethereum messages can be created either by an external entity or a contract and can contain data. Also, ethereum messages are type of a contract account and can return a response.Ethereum transactions are the signed data packages that store messages to be sent from an externally owned account. They contain the message recipient, sender signature and amount of ether (startgas) and data (gasprice) to be sent. If the transaction runs out of ether all state changes are reverted and if the transaction execution halts with some gas, remaining fees is refunded.

Architecture and Implementation:

Approach 1:

Architectural Components (Fully decentralized):



Sole component (multiple): Blockchain Network Node(s) with the deployed (ride booking) contract.

Decentralizing the application means to completely eliminate application dependency on any untrusted service. To achieve this, every user - rider/driver has to participate in the blockchain network as a node. The business logic of transactions related to Ride request, Quotations, Rider details, Driver details (including current location) is boiled down into contracts - an interface to interact with blockchain and store related transactions. These contracts are deployed in the network (with a unique identifier). All the nodes on the network are ethereum accounts whose permissions are set for specific ABI usage (in the contracts). So, on a transaction, let's say on a ride request by a node on the network, a consensus is achieved for creating a transaction to store on blockchain. This has to be picked up by the driver nodes on the network and the contract logic is developed to select the nearby drivers. These driver nodes respond by creating quotations (as transactions of a contract) which in turn are verified by all other nodes on the network. The communication between the nodes (business logic communication of notifications related to requests) is implemented into the contract.

(Note: We have tested this implementation by deploying small contract with basic Ride_Request and Quotation logic. The complete flow of registration, polling for requests was not tested and

not feasible - discussed later)

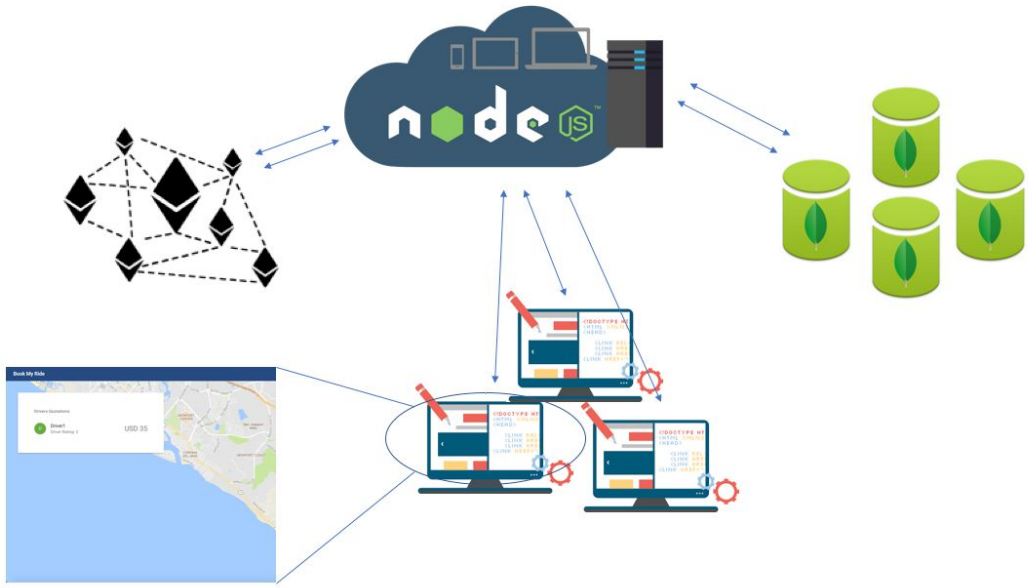
This solves the problem of complete decentralization as the complete logic of contract is held by all the participating nodes and the distributed-ledger storage helps the information storage. But this comes with a very high price.

Limitations:

1. Assuming that all the users - riders and drivers to participate in the consensus mechanism and distributed ledger storage is completely unacceptable.
2. Consumes very high resources - storing the transactions over the blockchain network seems fine until the details are not listed for example, storing continuous spatial location information of drivers to huge, polling for rides by drivers, polling the rides consumes huge amount of gas (ethereum gas - unit for computational expense)
3. Effort of development is extremely high - Currently, the contract logic is written in Solidity which can be compiled and deployed on Ethereum BC network is in primitive stages. There are no library implementations or any data structures available for developers. (We had to implement linking of all Quotations to a Ride Request with the implementation of a Linked list developed by us). The contact ABI does not support retrieval of structured data as well - all the information is stored using very primitive data types (does not include float but has a fixed point implementation).

To avoid these limitations, the application itself is not decentralized, but the business logic of transaction is. This allows thin clients to interact with blockchain network without forcing it to be part of it and interacting with it for transparency and verifiability.

Approach 2 (for Scaling and Realistic scenario):



Architectural Components:

The components included in this architecture are:

1. Ethereum Blockchain Contract (programmed in Solidity): A distributed-ledger used for transparency information - the transparency is exposed via REST api hosted by Co-ordinating server. The contract incorporates the business logic of a Ride (Booking) request, each of which contains the information related to the booking which include Ride ID, Rider ID, Source and Destination locations, List of Quotations proposed by the drivers (geo-spatially located nearby rider) - implemented as a Linked List. The Quotations themselves can be contracts (we implemented them as structs in the Booking contract itself) which contain transactional information related to a quotation by the driver : Quotation ID, Quotation amount etc.,

Ethereum smart contracts provide very convenient way of deployment. The key things a node in the network required in participating / interacting with the contract is the ABI definition and the contract address. This makes it simple to create instance of contract and interact with it directly providing the decentralized approach. The stored information is blocks of transaction verified by the whole network making it available for all the nodes.

2. Co-ordinating Server (Node.js): A Node.js server using express libraries for easy REST API setup. This server (single threaded for event handling which uses multi-threaded asynchronous implementation) as the name suggests coordinates the interaction between Ethereum Blockchain network, Rider Application (web thin client - no storage / nor computation), Driver Application (web thin client - no storage / nor computation) and MongoDB (that stores all the information - majorly chosen for the support on geo-spatial queries). The APIs hosted on this server provide the interface for users (riders and drivers) for interaction.
3. MongoDB: Database used for storing the information and supports geo-location based queries which helps coordinating server find nearby drivers for a ride request
4. User Application - Rider Web App and Driver Web App: React based front-end application that provides UI for the whole system

Approach:

Note: To reduce the complexity of the project, the communication protocols chosen for any interaction are chosen to be basic REST api and does not involve any message queuing nor notification systems.

Our implementation is limited to the registration and booking flow of a complete cab service: Creating ride requests, polling for nearby drivers (and ride requests), polling for corresponding quotation amounts and selecting a specific rider quotation, fetching the default price between two locations and confirming the ride.

For the scope of the project, for development, a virtual blockchain network is created using “ganache-client” which provides 10 ethereum virtual machine nodes that participate in the

consensus mechanism for distributed-ledger based storage. A coordinating server is then setup which deploys the Booking (Booking.sol) contract on the blockchain network. The server uses one ethereum account (default account 0 in our case - by creating an instance using the deployed contact's address and the contract's ABI) to invoke any interaction with the blockchain network. Apart from interaction, the server hosts REST API that provide functionality to users. This includes: 1.Registration API for riders and drivers to participate in the cab service 2.Rider API to create new requests, poll for quotations, confirm ride etc., 3. Driver API to create quotations for ride requests, updating driver location, confirming selected ride etc., and 4. Transparency API to verify the transactions involved during the whole process. All the information being coordinated such as requests, responses, locations are stored on MongoDB which is used heavily and easily for the purpose of Geo-location based queries (On a ride request from a rider, the nearby drivers). Currently only the critical information related to ride requests and the corresponding quotation details are only stored on the blockchain network. The notification scheme chosen for implementation is simple and is based on continuous polling i.e., continuous API requests to achieve notifications such as nearby driver list, corresponding quotations, driver selected status and confirming rides. Transparency API provide the functionality to verify the critical details in the transactions involved (Currently, we support only API based verification which include verifying ride request details, all the corresponding and related information for the quotations associated).

Note:

1. To achieve the default price between two locations, our architecture assumes a trusted parted such as Govt to do it for us by a REST API. The coordinating server gets this information from the trusted source to the user and the user can verify the same with the globally exposed API.
2. To achieve multi-riders booking the same driver (since it is more probable to happen as choosing is not server driven), we have implemented a basic spin lock mechanism that deals with concurrency issues.

Conclusion and Future work:

We have concluded that decentralized applications are the way the future applications are going to be designed. This project proves that, an application that even involves sensitive information can be stored on blockchain (with privacy parameters set). Also, since the current consensus mechanism being used (PoW) for storing transactions, a complete decentralized approach might be a little too dreamy (Approach 1) but hybrid mechanisms (Approach 2) definitely provide concrete results meeting realistic scenarios. Quantitative evaluation scheme is not used here but since the project uses technologies which are currently industrial standards, we can safely say that blockchain layer adds up transparency as a bonus functionality.

The cost of computation might be reduced by using different consensus mechanisms like Proof of Stake - being incorporated into Ethereum which will make decentralized applications as an industry standard meeting everyone's goals.

The current project implementation does not answer the complete cab service solution. It is a demonstration that blockchain can be added as transparency layer or can completely change

the architecture to decentralized mode. These are the possible additions to extend our work and make it more complete: 1. Complete all flow cases - A ride completion flow, 2. Implement the service on EVM (Ethereum Virtual Machines) n/w rather than virtual n/w, 3. A completely decentralized flow utilizing only DB (eliminating coordinating server) for location information and geospatial queries 4. Develop a faster and better way to do geospatial queries on blockchain itself - utilizing the map kind of storage 5. Alternate decentralization approach based on spatial locality and many more.

Design specifics:

Ethereum Blockchain Network:

The network is a virtual blockchain network created by ganache client with 10 nodes. The contract deployed on this network is Booking.sol. The transparency of the blockchain is exposed via REST api hosted by Co-ordinating server which are as follows:

Blockchain APIs:

1. Getting all quotation details of a request: /api/v1/getAllQuotationDetails
2. Getting all request details of a request : /api/v1/getReqDetails
3. Total Requests: /v1/getTotalNumReqs

Contract structures:

```
struct Booking_request{
    bytes32 request_id;
    bytes32 rider_id;
    bytes32 source_lat;
    bytes32 source_lon;
    bytes32 destination_lat;
    bytes32 destination_lon;
    bytes32 chosen_quotation_id;
    bytes32 quotations_head;
    uint8 num_quotations;
```

```
struct Quotation{
    bytes32 quotation_id;
    bytes32 driver_id;
    uint8 quotation_amount;
    bytes32 next;
  }

uint8 public num_booking_requests;

 mapping (bytes32 => Booking_request)
```

mapping (bytes32 => Quotation) quotations;
 	
}
 	
	public booking_requests;

Coordinating Server - NodeJS:

It is the coordinating server hosting the rider and driver application functionality. The following APIs exposed by this server:

- Rider APIs
    1. Registration of a new rider: **/api/v4/register/rider**
    2. Creating a ride request : **/api/v1/rider/createNewRequest?rid=x**
    3. Poll quotations for created ride request: **/api/v2/rider/pollQuotations**
    4. Select final quotation - for a rider: **/api/v2/rider/selectRide**
- Driver APIs
    1. Register Driver - **/api/v4/register/driver**
    2. Driver update location - **/api/v2/driver/updateLocation**
    3. Poll nearby rides for a rider : **/api/v2/driver/pollRides**
    4. Add a quotation for a ride - by a driver - **/api/v2/driver/addQuotation**

MongoDB

- MongoDB holds information corresponding to the following collections:
    - Driver
    - Rider
    - Ride Request
    - Quotations
- Allows geo-location based queries and retrieve further information.

The collections structure is as follows:

- Drivers:{ _id, driver_name,status,driver_rating,driver_car, curr_location}
- Riders:{_id,rider_name}
- Ride_Request{ _id, rider_id , src_lat , src_lon, des_lat , des_lon, Quote}
- Quotations { _id, driver_id, req_id, quote_amount}

References:

- Dixon, Chris. ”Why Decentralization Matters Chris Dixon Medium.” Medium. Augmenting Humanity, 18 Feb. 2018. Web. 12 May 2018.
- J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In Advances in Cryptology - EUROCRYPT 2015 - 34th Annual InternationalConference on the Theory and Applications of CryptographicTechniques, Soa, Bulgaria, April 26-30, 2015, Proceedings,Part II, pages 281310, 2015.
- Elias Baixas, Satya, Gopal Malyala, James Lyndon, RossDemush, Bernd-Axel Hugelmann, Neeraj Murarka, AdrianLazar, Adrian Lazar, Amrik Mahal, William Tucker, AlecChalmers, Larry Jackson, and Gunnar Forsgren. ”Proof of Work vs Proof of Stake: Basic Mining Guide.” Blockgeeks.N.p., 28 Dec. 2017. Web. 12 May 2018.
- Byzantine Fault Tolerance.” Wikipedia. WikimediaFoundation, 11 May 2018. Web. 12 May 2018
- Luu, Loi, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena,and Aquinas Hobor. ”Making Smart Contracts Smarter.”
- Proceedings of the 2016 ACM SIGSAC Conference onComputer and Communications Security - CCS’16 (2016):n. pag. Web.
- Ethereum/wiki. A next-generation smart contract and decentralized application platform, October 2015.
- Node Etherium dapp integration - https://medium.com/coinmonks/node-js-backend-service-for-ethereum-dapp-sotp-part-3-2d3aa5ec50e9
- Node MongoDB driver documentation - http://mongodb.github.io/node-mongodb-native/2.2/
- Node Async-lock package documentation - https://www.npmjs.com/package/async-lock
- Truffle testing framework, Ganche client - https://medium.com/coinmonks/testing-solidity-dapp-sotp-part-2-9685b3375aaf
- Solidity programming documents - http://solidity.readthedocs.io/en/v0.4.21/