Abby Mueller and Jared Joyce
Distributed Systems
12/16/23

Real Estate Blockchain

Introduction

Our project seeks to implement a blockchain for real estate transactions. The concepts behind blockchain are believed to have first been proposed by David Chaum in 1982[1], but rose to more popular knowledge with the advent of Bitcoin in 2009. Blockchain has many benefits, including a decentralized structure, security, and immutability. Although blockchain is commonly associated with cryptocurrencies ("crypto"), its ability to act as an immutable ledger also makes it appropriate for certifying transactions such as that of real estate. The adoption of cryptocurrencies in modern finance has been rapid and lucrative for many. Blockchain (and crypto) also provides a level of security for those without stable currencies or with corrupt governments. We believe that in the future the blockchain can be used for things beyond currency transactions. We believe that some of the properties that make blockchain great for currency transactions also can have wide applications from diplomas to real estate. In this project, we develop a real estate blockchain designed to function on large systems to help serve as support real estate market in the United States.

Related Works

In addition to reading various blog articles about blockchain and blockchain in real estate, we also found it useful to consult research papers and scholarly works about blockchain.

While Bitcoin[2] deals with blockchain in a cryptocurrency sense, a lot of the ideas about blockchain are not limited just to cryptocurrencies. It is considered a foundational paper by most in terms of blockchain, and

[1] Citation https://nakamotoinstitute.org/static/docs/computer-systems-by-mutually-suspicious-groups.pdf
[2] https://bitcoin.org/bitcoin.pdf

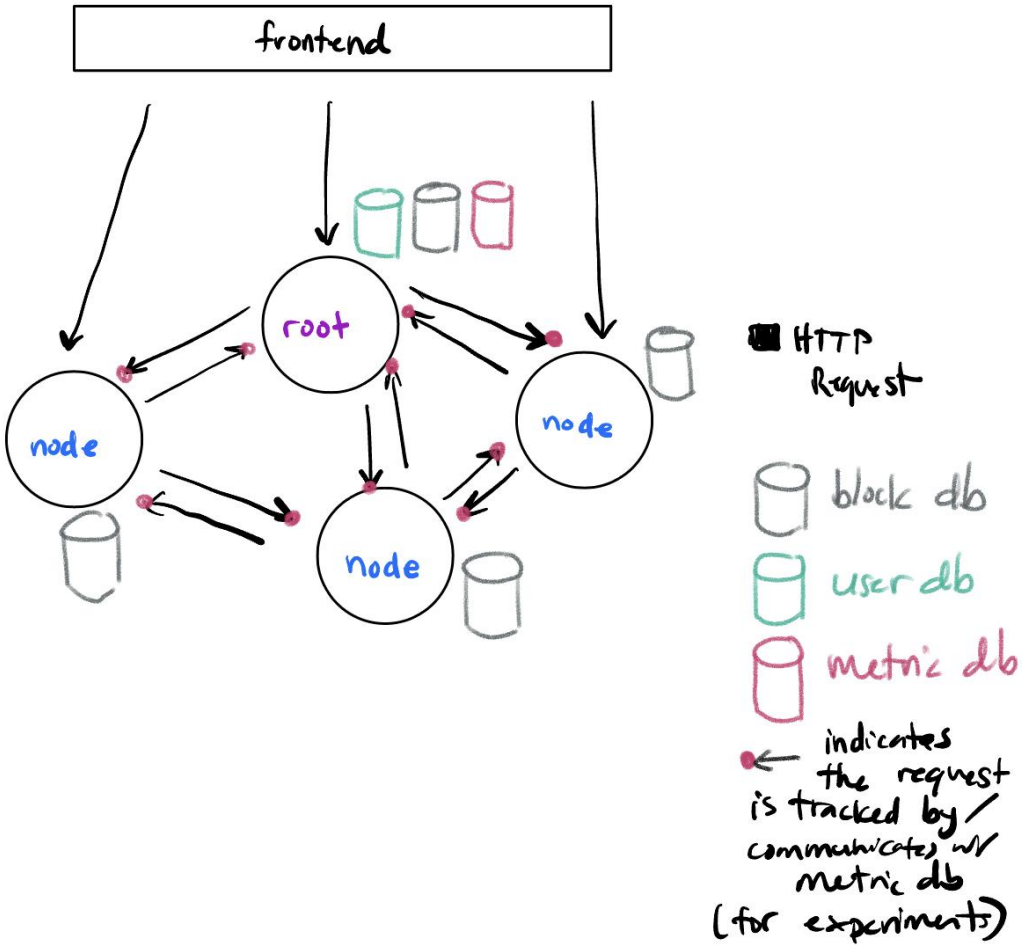
we treated it as such for our project. Key ideas present in Bitcoin that we also use in our system are blocks, transactions verified by a hash and previous hash, generating hashes with a nonce, running a network utilizing broadcasts/messaging to other network nodes, and using the longest chain as part of the consensus algorithm.

The SimBlock blockchain simulator paper[3] was also useful: this was a paper for a system that was designed to simulate various blockchains in order to run experiments on them without having access to many nodes in a WAN. It was useful to think about common pain points of blockchain projects, like transaction throughput. We did test our blockchain with actual nodes/requests unlike this paper, but it was important to think about how this might be difficult to access, especially as the number of nodes scaled even larger than the ~40 something-50 machines we had access to as part of this project.

It was interesting to note that the majority of papers dealing with blockchain did not deal much with experimental results sections or graphs detailing experimental results; they mainly concerned themselves with explaining the system's design and how certain features (security, etc) are supported by the design/specific implementations.

[3] https://arxiv.org/pdf/1901.09777.pdf

Design/Architecture Fig 1: Architecture Diagram of System



We designed our system to be a network of peer-to-peer nodes. Each node has the same functionality, and the system can add new nodes through an HTTP request sent to a known node on the network. We designated one node the root for the sake of having the first node join the network, but the root doesn't have much different functionality. Its uniqueness (other than being started up first) is the databases rooted there. We are rooting our user database at the root node as we do not need a separate user database for each node (users are common across the network), but it does need to be somewhere in the network. The same goes for our metrics database.

The user database exists in order to better control access to front end (and, specifically, access to node-level functionality) and to help with security and resource control. Once a user is logged in they can

accessed the front end. In the future we wish to add node ownership as a logged-in property so a user doesn't have access to all nodes. The metrics database is useful for evaluating our system's performance.

Each node stores a blockchain, made up of various blocks. Important information for the blocks include a timestamp and transaction information. Timestamping is important for settling cases where a transaction may be being mined on multiple nodes.
The nodes have block databases to allow storage to be larger than just a data structure. This allows also for easier, faster lookup (instead of iterating through a list of pending transactions, we can do a database lookup, which has a faster average performance).

Our nodes (and blockchain related operations) are accessed/controlled from a front-end that allows a user to see pending transactions, try to purchase properties, and initiate mining or terminate their own machines. For the sake of testing, we wrote automation scripts that directly made calls to endpoints, but we have a very user-friendly UI for individual users!

Key parts of our design, like most blockchain, include a broadcast when a node joins the network, adding transactions, as well as mining.

Where our system is different from other blockchain like Bitcoin is that new transactions are sent to N nodes not all nodes. We hope that, while still allowing for fault tolerance, this may reduce overall network traffic. Our experimental results help show this. Additionally, if a user feels really strongly, they can choose to still replicate to all other nodes to minimize some of the tradeoffs from very

Implementation

Technologies

To implement our blockchain we used end-to-end Javascript. Specifically, we used Vue with Typescript for the front end and Express.js with Typescript on the backend all running on Node. For our database, we used MongoDB and interacted with it using a Node package called mongoose. We used docker for containerization (and bash scripts for automated deployment/destruction of these containers) and we used AWS for testing across different machines and locations.

Databases

Our Backend had 5 different database models across 3 major classes. 2 of those databases were shared across all the nodes and stored on the root node. The 2 databases we did not replicate were the user and metric databases. The user database stored user information including username, name, password hash, salt, and associated nodes. In a larger-scale system, we would prefer to have the user database replicated but to test our system and basic functionality it did not seem necessary. The other database on the root node was the metric database. All nodes have middleware that sends network information to the metric database to collect performance information.

Each node stores its own blockchain database. This database can be split into 3 models: Transaction Data, Block Chain, and Peer Nodes. The peer nodes, as the name suggests, store information about the peer nodes across the database. This information would be important as the blockchain network grows to large sizes. The Transaction Data model stores all the information about the given transaction including timestamp, price, uuid[4], previous owner, new owner, address, and its pending status. The Block Database stores the block information document on the block database containing information about each block including nonce, previous hash, data (references to transaction documents) timestamp, and index. Unlike normal blockchain implementations using references to transaction information allows us to leverage Mongo's lightning-fast querying speeds for up-to-date transaction lookups. We also created the

[4] Unique user identifier

previous hash out of the transaction data and not the document to account for the changing transaction document not creating issues. This ensures the security of the blockchain.

Sending a new transaction

When a node receives a new transaction first it runs checks to make sure that the address isn't currently a local pending transaction. After, it stores the transaction on the database with the pending section set to true. Depending on the replication environment variables it will send N requests to other nodes on the blockchain asking them to replicate the newly received transaction. This ensures that new transactions are replicated even before any mining is guaranteed. Once this is done the transaction just waits for a new block to mine the transaction.

Mining a new Block

When a node receives a request to mine a new block, it first awaits responses from all other nodes in the blockchain to give the node their pending purchases. This implementation ensures that if there are pending purchases it will be included when a block begins to mine. Once the block is gathered the node runs additional validation on the transaction (2 transactions can't be bought at the same time). Once the node has all the valid pending transactions it begins to run a consensus algorithm. The consensus algorithm is pretty standard: the node uses the longest valid blockchain as the true blockchain. To check the validity of the blockchain the node ensures correct previous hash information by rehashing the previous block using sha256 and checking to see if the nonce is valid. After the most up-to-date block is found, the block is mined using normal Proof Of Work protocols: first, the block finds a valid nonce, then it creates a valid hash then it stores all the information in the block document by adding all the transaction document IDs followed by changing the pending status.

Evaluation

We evaluated our system running on 16 nodes and 25 nodes by running various frequencies of purchasing and mining requests from dataset of 1000 properties. We generated larger datasets but for the sake of not tying up the shared cluster of computers, we decided this was a large enough number to not be insignificant but small enough to run quickly even when varying multiple variables. These nodes were also located in different geographic regions globally.

We changed how many purchases were submitted before a mining request was made (100, 25, and no mining) as well as how many replications of a transaction were specified– we tested 1, 4 n // 2, and n -1, which were a few small data points, about half replicating, and full replication. Our figures are below, followed by analysis. We used “no mining” as more of a control as this system with no mining could not verify property purchases, which is part of its purpose.

Fig 1: Size of payloads being sent across the network comparing varying mining frequencies and number of specified replicas of transactions

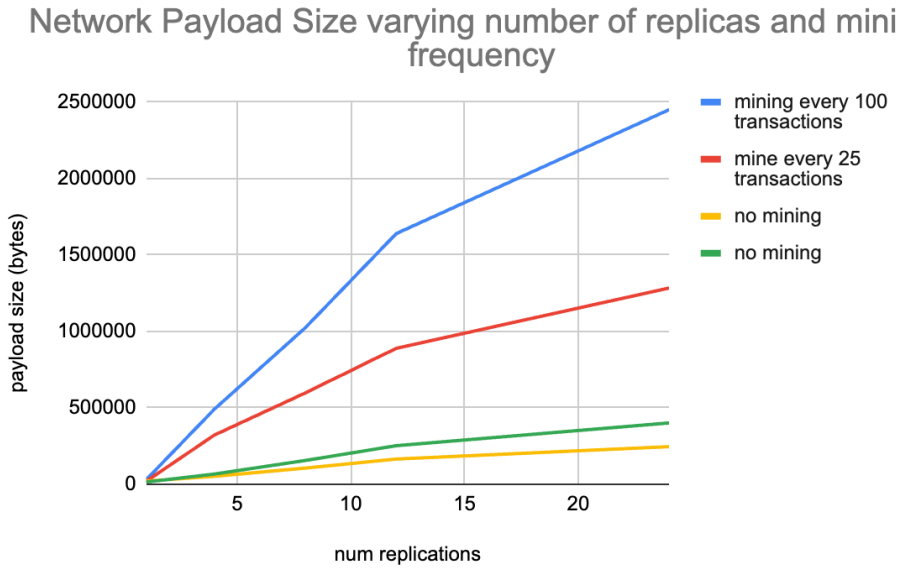


Fig 2: Number of requests being sent across the network comparing varying mining frequencies and number of specified replicas of transactions with 25 machines

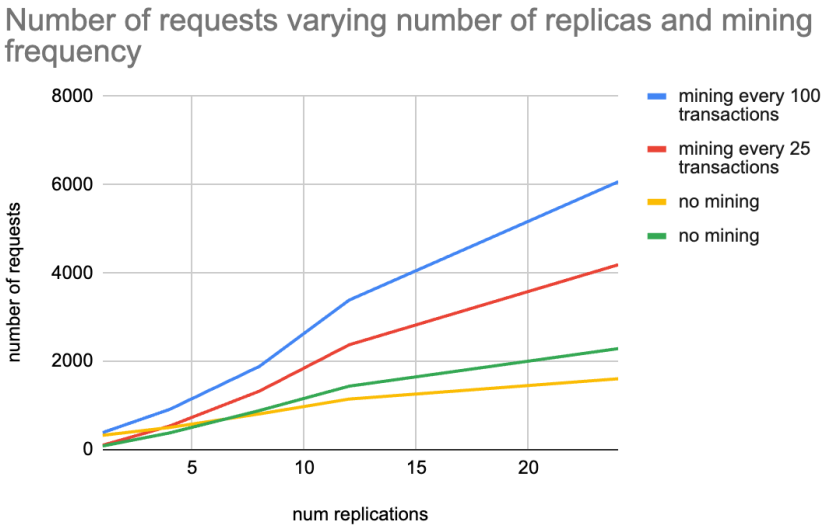


Fig 3: Size of payloads being sent across the network comparing varying mining frequencies and number of specified replicas of transactions: 16 machines

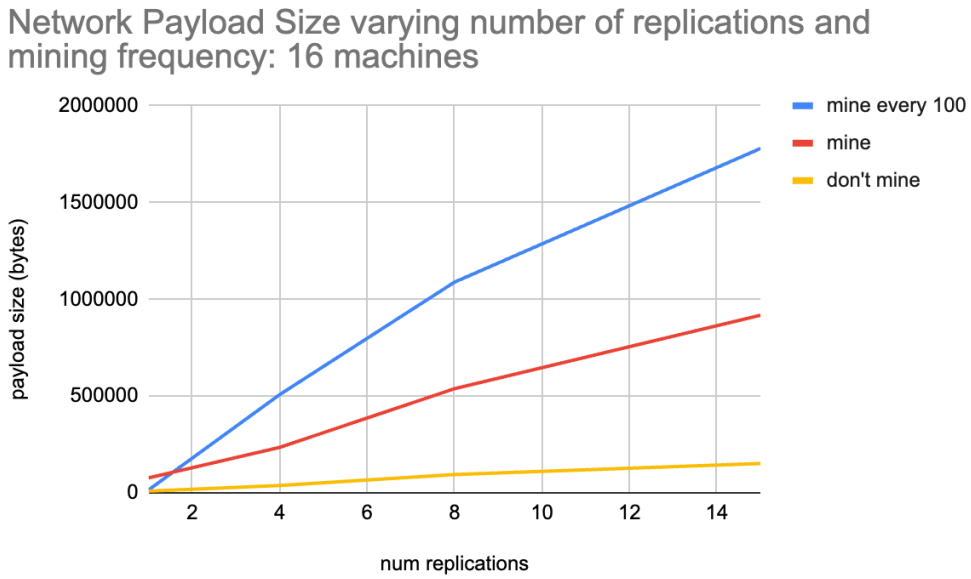
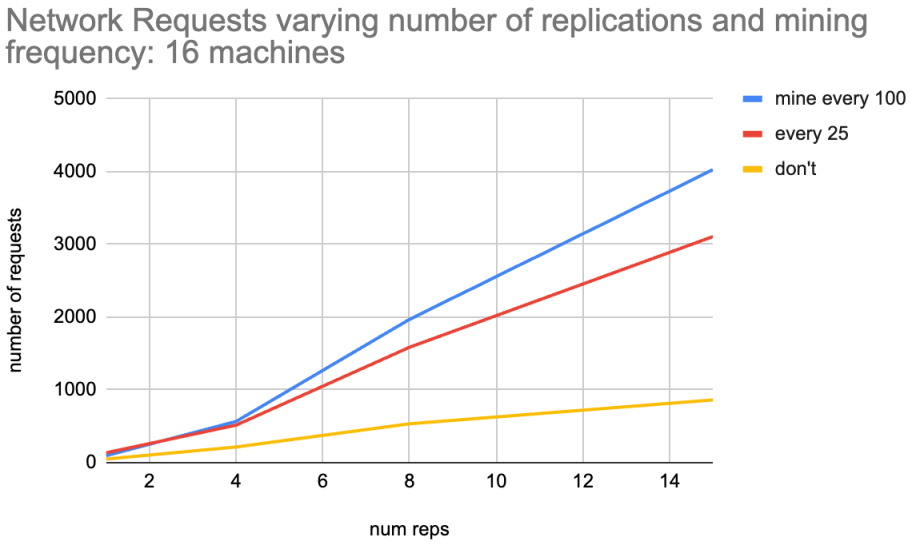


Fig 4. Size of payloads being sent across the network comparing varying mining frequencies and number of specified replicas of transactions: 16 machines



Something that sets our system apart from typical blockchain is that we allow the number of replicated transactions to be specified. There are some tradeoffs here– too few replications and you run the risk of a transaction being lost if mining is not frequent (fault tolerance lost), but these data help show some of the benefits: less overall data being sent around the network outside of mining, reducing total network traffic.

We collected our data using our metrics database (middleware), stored at the root, which tracked number of requests. In terms of Fig 1, discussing payload in the network, this was greatly inflated by mining, which we expected given that mining results in the sending of new blocks verifying transactions to other nodes on the network as part of consensus. We tested differing frequencies of mining (as well as what network traffic looked like without mining). We did two different iterations of non-mining tests due to odd network traffic in the cluster of computers provided to the class at the time of this experiment (several machines were slow/down at the time of the first experiments). This was the same in figure 2.

This data confirmed our initial hunch: as there was more mining, the overall network traffic as measured by payload increased, as with more replication of transactions as well. The overall payload in the network was also greater with more machines (25 vs 16), which made sense (See Fig 3.: the max payload was 1779832 bytes for 16 machines with most frequent mining versus 2450950 bytes with the same mining frequency for 25 machines).

Discussing figs. 2 and 4, the number of requests, is also very interesting. As we mined less frequently but with more pending transactions (every 100 transactions vs every 25), the overall number of requests was still greater, most strikingly as the number of transaction replications was increased. The increase in requests as replications were increased made sense as replication increases requests on the network, and we believe that mining less frequently meant there was more to settle regarding consensus, yielding more request traffic as nodes communicated.

As a caveat, our number of requests was not what we expected. We sent out a purchase request for each of the 1000 properties, but they were not all counted in the middleware. We attributed some of this to network issues and/or lingering synchronization bugs, especially given that all request counts and payload quantities needed to be stored in the one metric database. We suspect this was mostly network issues or middleware bugs. Given the slow throttling of network requests to our blockchain while gathering data (1000 requests took between ~3-5 mins to send, atypical especially in cases where we were not mining blocks, which should have resolved this issue by speeding up request processing time, as purchases do not require the same rigid proof as mining and network validation).

However, since we ran all of our experiments under these same conditions, we do think the trends reflected in our graphs accurately reflect the increase in network constraints imposed as the blockchain gets bigger in both size of blockchain and number of nodes in the network.

Further Experimentation

We think an important experiment to run in the future would be measuring purchase query time, to quantify the benefits of using the Mongo database as the blockchain grows and more purchases are made. However, we had already implemented our system using the database and would have needed to construct some sort of contrived dummy lookup function or have tested query times before we did the implementation work, both of which would have been/are possible but were out of the scope for this paper.

Conclusions

We successfully implemented a blockchain for real estate, allowing for security and immutability. We also considered ways to reduce overall blockchain network traffic while still allowing for some fault tolerance with pending transactions before mining occurs, as most blockchain eventually become limited in terms of throughput. There are further areas we can go with this project, but we're excited with how our implementation and project turned out (and Jared is especially excited about the Vue frontend).

Other Citations

Other than papers referenced from footnotes, we did employ some use of ChatGPT specifically for making Python and bash lists of IP addresses as well as figuring out how to write a bash script/debug bash. As well as some general debugging/error management support.