Veracity Blockchain

Hayden McAlister

In this research project we investigated how distributed ledger technology could be used in a veracity context to verify claims and reach consensus on the state of a system. Other research was done concurrently into using Prolog to describe the state of a system and look at provenance in an organ donation scenario. I built the infrastructure to integrate this Prolog into a jupyter notebook (for accessibility and easy interaction) and expose the blockchain for use.

We agreed that for a proof of concept a small scale, locally hosted blockchain would be most suitable. There is no need to put our tests on a "real" ledger at this stage, but if this project moves forward in this direction it may be useful to put information on a public ledger so it can be verified.

My first task was to find a suitable blockchain technology that could be locally hosted. I found Hyperledger to be very promising, as the opensource projects could be locally hosted and seemed very flexible to cover many situations. I looked in to using Hyperledger Indy, Aries, Besu, and Ursa but determined that Hyperledger Iroha would be most suitable for my needs. This is mainly due to Iroha having the best documentation of the projects I looked at, and was the easiest to start developing with quickly. However, other projects could absolutely be used and in future local hosting may not even be needed.

I spent several weeks setting up infrastructure using Iroha and Docker. I managed to get four Iroha nodes running and communicating in separate containers, which is significant as four nodes is the minimum necessary to detect and correct for a single faulty node (crash fault or byzantine fault). While this demonstration does not look any further into faulty nodes, this infrastructure would support probing in this direction.

Because the goal of this project was to eventually host some Prolog work, we had to get Prolog running in a jupyter notebook. Thankfully this task was done for us (and can be found here), and with some scripting to copy kernels around containers, we have a jupyter notebook that can run Prolog code! The kernel essentially runs Prolog line by line through a python interface with SWIPL, then dumps the factbase into a file for consultation later. These files are referenced by Prolog when querying the factbase, so the files store the state of the world. This will be useful later.

The kernel, while fantastic, needed extending to work with the blockchain network. I added some extra functionality which made development of Prolog with the blockchain a little easier. I added functionality for:

- An interface to the Iroha network, for asserting the hash of a fact file onto the chain and querying the chain for the existence of a hash
- An interface to a redis database, so a given hash can be resolved to the fact file that generates it. This allows for consulting of hashes to create a copy of a factbase
- Timestamping of fact files, so the same fact file can be committed multiple times onto the blockchain (this is mostly useless but *very* useful for testing)

A walkthrough

Finally, we have seen enough of the infrastructure that we can get a broad overview of how this project works.

- At startup of the notebook (and hence the kernel) the user blockchain data (private key) is loaded and an interface to the blockchain is created.
- A Prolog cell is created and run, defining some facts about a system (the local kernel believes these facts but a different, remote kernel does not even know the cell exists)
- Upon running the cell, the SWIPL kernel timestamps the file (adding the current time to the start of the file) and hashes the file. The resulting file hash is stored on the blockchain.
- The kernel then stores the file hash in the redis database, mapping the hash to the file itself.

These final three points can be repeated any number of times, creating any number of files and adding these to the factbase. After the local user has finished creating files, we know all of these file hashes are stored on the block chain and the file hashes are mapped (via redis) to the original files. If the local user then runs some arbitrary query on this factbase, they will get some deterministic result. Exciting! But so far nothing exceptional has happened.

Now consider what happens if a different user (User B) wants to verify the original users (User A) query. This other user must know the entire factbase of User A before running the query, but asking User A directly could result in compromised communication or corruption somewhere along the path. User B could therefore:

- Start the kernel, loading User B login data and creating an interface to the blockchain
- Query the blockchain for all hashes commited by User A
- With this list of hashes, User B can consult the redis database to get the matching factfiles
- User B can now run the query with the same factbase as User A, getting the same results

Of course there is still some issues with this method. Communication could still be compromised or corrupted between the blockchain or redis database and User B. The redis database could map from the hash to an incorrect file (particularly if User A is malicious in storing the file hash). The process overall is much more complicated.

However, the upsides are very important! Because the blockchain is queried rather than an individual user we can be sure that *any* user will get the same query results, while if User A is queried they could change answers based on who is querying. If encryption of files is implemented in the redis database then we can be sure that the correct file is stored in the data base (i.e. the file is encrypted by User A so must be decrypted using their public key, so User B can be sure the file is both stored by User A and has the correct hash). Finally, we should consider involving more than two users. If we had several users, all considered authoritative on some selection of facts, we could suppose that each user may only commit facts that they are authoritative over. Each other user could then collect the committed hashes and apply only those files with facts from authoritative users. This would mean we have a system where some users can set some facts and other users other facts, but *all* users can see *all* facts and agree on them.

As an example, in an organic supply chain we could state that a farmer could assert facts about their farming practices, a transporter could assert facts about transportation, a supermarket could assert facts about storage, and a consumer could collect all of these facts to determine the organic status of a product. Because these facts are committed to the blockchain all users can be sure that their factbase comes from authoritative sources, all assertions are public and agreed upon, and all users have the same factbase.

Summary

This project demonstrates that distributed ledger technology can play an important and powerful role in the veracity space. However, the blockchain should be augmented with other tools (here, Prolog) and it is important to understand *why* the technology is useful. In particular, this project shows the usefulness of the distributed consensus and append-only properties. While blockchain technology can form a useful basis for veracity applications, it should be combined with other technology or concepts to be truly useful. A blockchain on its own will not solve the veracity problem.