Summary of understanding and a proposal to evaluate Fat Contract use cases

I think finding a good use case starts with a rigorous understanding of the product at hand and its competitive advantage over its competitors. We can't simply name processes that suffer from some ill-defined data privacy problems, and propose to push those very processes into the "blockchain", because doing so puts the term "blockchain" into the abstract and ignores the particular motivations/features of the blockchain we are dealing with.

I propose the first step as this: to write down, fully and clearly, the outstanding product features of Phala's fat contract. After that, we can develop some mental framework (checklist or so) to evaluate if a use case is good or not. To do so, we also need to know what a smart contract is precisely. Hopefully reading this can provide some clarity; writing this certainly helped me.

A smart contract is simply, a program that runs on the blockchain that delivers some promises. On Ethereum, it is an account (yes, think bank account). Think of it as a vending machine, Chris slides a dollar in and the machine drops a Cola, i.e. it has a protocol that can be coded up in some Python if-else statements for that matter, and it delivers a promise. Moreover, a smart contract has its own balance (remember it is an account), just like a vending machine has its own coin balance. A smart contract is immutable, meaning it cannot be destroyed once deployed, and it is owned by no one (simply deployed to the network). This makes us instantly think, what if there's a bug in the program? What if I want to upgrade the program? There seems to be no way to change it without fully deploying a new one. Does fat contract tackle this problem? More importantly, you may ask then, who gets to "own" or earn the money in this vending machine? The answer is written in the smart contract. The protocol (a fancy word for the algorithm) may be to transfer all balance at a regular time interval, perhaps every month, to another Ethereum account, which happens to belong to a guy called Chris, who deployed this smart contract. However, Chris doesn't own the smart contract, he simply defined it.

Well, you might ask, what's the point of a smart contract at all? (actually pretty dumb, considering it just executes an algorithm lol) In real life, contracts are enforced by the state, a third party. If American airlines didn't deliver flight delay compensation as promised, the police might come to poke around their offices the next day, i.e. it breached the contract. However, this is extremely costly considering the police/lawyers/courts that are involved. What if contracts can be enforced digitally? What if no third party is needed? This is the core motivation of a smart contract, that the contract's promises aren't guaranteed ultimately by a third party (the state) but by the code. It is one reason why a native currency (i.e. ETH) is needed because the code can only guarantee on-chain transactions but not the 5-dollar bill my mum gave me to buy cola in real life.

Smart contract also comes with two added benefits: 1) zero ambiguity, i.e. can't hire a million-dollar lawyer to "re-interpret" the contract since it's all computer code, and 2) transparency, i.e. anyone can read the terms of the contract and validate the transactions executed under the contract.

Now, Ethereum's smart contract can't fit all use cases. Not because it's flawed, but because of its specific design. It is optimized for resilience, not performance, resilience surrounding a single, shared truth that cannot be tampered with or changed. Quoting the words of a stack exchange answer, "It is designed for a hostile environment in which up to 49.9% of the nodes may be not merely non-functional (crashed), but overtly hostile and deliberately sewing confusion in the network". As long as the majority of the nodes are honest (which becomes highly likely as the network grows in the number of nodes), this single truth can be guaranteed.

Correspondingly, Phala is not happy with the three design features of Ethereum's smart contract (or any other legacy blockchain's smart contract) as followed. It might optimize for resilience, but not other things. A good question to have in mind, then, is does Phala compromise some resilience in its design?

1. All data is public. Being a public blockchain, all data is visible. So anyone can see and verify the transactions. This means Haechan can verify that Chris bought 10 colas from the smart vending machine if he wishes, but I wouldn't like people to know that. Being able to see the data also means there is a possibility to hack it and change it, called the 51% attack, though highly unlikely.

2. A lot of mining work is wasted. Being a permissionless blockchain, anyone can be a miner. So multiple miners might find themselves processing the same transactions and competing to publish a block, but only one miner's block is accepted into the network. Though, the other's work wasn't wasted due to a design fault. Recall Ethereum is built for resilience, the extra work is necessary for the fundamental assumption of trustlessness that I as a miner, cannot trust 1) a central authority to fairly assign me a block without ulterior motives, nor can I trust 2) so many miners to be so kind/respectful as to take turns to mine blocks. Ethereum, or blockchain/ cryptography technology in general, supposes a state of anarchy where no one can trust anyone else.

3. External events cannot be queried. Being a decentralized network, smart contracts are executed independently by every node on the chain. Why? If a smart contract is executed by only one node, say Vitalik's computer in Malibu, and he hands the result of the execution to all the other nodes, then blockchain is a hoax. He can change anything and I'll believe the vending machine gave me 10 colas, not 2 in reality. This is the fundamental doubt cast on centralized cloud computing - your program hosted on AWS could be hacked in Virginia (since it's centralized) and show faulty results without anyone using it knowing. If all node executes the smart contract independently (even if some nodes are hacked), then we can verify that all our results match and the glorious truth is upheld. This is why external events cannot be queried. If all 300,000 Ethereum nodes call Facebook's stock price API off-chain, how can we make sure the API response, the stock price, is the same (it's not) every time and the glorious truth is upheld? With latency (a fancy word for the delay), every node will get a different truth, which is not good. So what's the workaround? We can trust an oracle database to call the API once

(which is what people actually do) and share it with every node in the blockchain. But that instantly defeats the goal of trustlessness. We might as well use a regular SQL database.

Given Ethereum has data privacy issues and some serious performance issues, here comes Phala. The million-dollar question, again, is does it make tradeoffs with resilience surrounding a single truth? How does Phala address these problems? Here I refer to a Fat Contract as a program on the Phala blockchain environment, similar to a smart contract being a program on the Ethereum blockchain environment. Despite this making logical sense to me, I've seen Fat contract being referred to as the protocol that governs how the Phala blockchain works (I'm not sure about this).

Correspondingly, here are the pointers that I still need to gain clarity on:

1. All data is encrypted unless specified. No one on the Phala blockchain can see what the underlying data and transactions are unless specified, only the hashes of the data. Does it make it a private blockchain? How can Haechan see and verify that Chris bought 10 colas from the smart vending machine, if he wishes to (he can't if the data is encrypted/ not transparent, which is paradoxical)?

2. No work is wasted. Being a permissioned, instead of a permissionless, blockchain, all miners (or Phala calls it workers) are background-checked and trusted. Gatekeepers assign blocks without ulterior motives and workers verify transactions without the ability to hack because they can't even see the data in the first place! This is also achieved by technology like TEE and asymmetric cryptography between gatekeepers and workers, details of which are not important for our purposes. Assuming the background check (which seems a bit manual) is fool-proof and we can trust the workers, how scalable is the worker pool and hence the network with a barrier of entry? There's a reason why an Ethereum miner doesn't need permission from Vitalik and his core group of software engineers the cost of monitoring a pool of miners doesn't scale well.

3. External events can be queried easily. The documentation says, Fat contracts have internet access and can send HTTP requests. But are these off-chain queries (API calls) done by one node only or does every node do it independently? Because the crux is maintaining the single truth despite every node executing the fat contract independently (and making Facebook stock-price API call independently), not whether internet access is present (Ethereum's smart contract can have internet access too no?)

The answers to the above questions seem important to me because we need to know how exactly Phala's fat contract outperforms the existing option (i.e. Ethereum's smart contract or non-blockchain-based options like a database) while maintaining the resilience surrounding the single source of truth. Perhaps even before considering a fat contract use case, a more fundamental question would be: is blockchain even needed?

It seems so obvious but maybe Fat Contract is our hammer and we've just been hitting nails (use cases good or not). Below I proposed 1) a rough flow chart to consider a fat contract use case, and referenced 2) an online flow chart to answer the first question "is blockchain needed". They might help convince ourselves the use cases we have now are good (ad sense, stock-price prediction, voting), or pick other use cases more intentionally. I don't know.

A use case flow chart for blockchain:

1. Does it need a shared, historically consistent data store? If no, spreadsheet
2. Does more than one entity need to contribute data? If no, database
3. Are data records, once written, never updated or deleted? if no, database
4. Does it involve sensitive information that doesn't require mid-term to long-term confidentiality? if no (requires long-term confidentiality), encrypted database. blockchain is not good for long-term confidentiality.
5. Are there trust/control issues over who runs the data store? If no, managed database
6. Do you need a tamper-proof of all data records to the data store? If no, database. If you don't need to audit what happened or when it happened, you don't need blockchain.
7.

If yes to all the above, we may consider a blockchain use case.

Sources

- Point 2 on Eth – a lot of work (and CPU resources) is wasted: https://ethereum.stackexchange.com/questions/64058/how-ethereum-consensus-validates-same-transaction-from-different-miners

- Point 3 on Eth - smart contracts can't interact with external events / APIs: https://www.coindesk.com/markets/2016/04/17/why-many-smart-contract-use-cases-are-simply-impossible/

- Blockchain use case flow chart – page 42: https://nvlpubs.nist.gov/nistpubs/ir/2018/nist.ir.8 202.pdf

Christopher Fok – Feb 12, 2022