International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Viability of Smart Contracts for Large Scale Applications using Game Theory Concept

Costantine Paschal Kulwa

Blockchain Expert and Assistant Lecturer, Information and Communication Technology (ICT),
Tanzania Institute of Accountancy (TIA), Tanzania

Abstract: ***Blockchain smart contracts show the next stage in the development of protocols that support the interaction of independent nodes without the presence of a governing authority. Blockchain smart contracts are believed to be a potentially enabling technology for a wealth of future applications. But turns out, like every other maturing technology, blockchain smart Contract also has it challenges and limitations. Understanding these challenges and limitation can help business making decision before they put their efforts in blockchain application development. In this paper game theory is combined to look into what Smart Contracts are and what they are assumed to be. The aim is to give businesses a clear idea about smart contracts and help them to decide if the contracts are viable for large scale application.***

Keywords: Blockchain, Smart Contracts, Game theory, Business, Payments

1. Introduction

Over the past decade, blockchain technology has attracted tremendous attention from both academia and industry. The popularity of blockchain was originated from the concept of crypto - currencies to serve as a decentralized and tamper - proof transaction data ledger. Nowadays, blockchain as the key framework in the decentralized public data - ledger have been applied to a wide range of scenarios far beyond crypto - currencies, such as the Internet of things, healthcare, and insurance.

[5] Smart contracts are protocols defining self - enforcing, digital contracts. The main aim of such contracts is to guarantee fair exchanges between untrusted and independent entities. When smart people hear the term “smart contracts”, their imaginations tend to run wild. They conjure up dreams of autonomous intelligent software, going off into the world, taking data along for the ride. Unfortunately, the reality of smart contracts is more mundane (lacking of interests or excitement). The problem with smart contracts isn’t just that people’s expectations are overblown; it’s that these expectations are leading many to spend time and money on ideas that cannot possibly be implemented. It seems large companies have sufficient resources to travel a lengthy path from the moment when senior management encounters a new technology, to when that technology’s advantages and limitations are truly understood. Perhaps our own experiences can help shorten this time.

The questions is, are the smart contracts viable for large scale application? This is the aim of writing this paper and the discussion will be supported by game theory. Ethereum empowers developers to design and implement their own game theory systems in the form of smart contracts. Game theory mechanics are what make blockchain so special. Nothing about the technology or mechanics is new, but it is the marriage of these two fascinating concepts that has made cryptocurrencies secure from internal corruption. Additionally, I discuss the application of game theory on smart contracts and finally I highlight three important challenges that make blockchain use cases difficult to implement.

Understanding smart contract model

[5] A contract is an instance of a computer program that runs on the blockchain, i. e., executed by all consensus nodes. A smart contract consists of program code, a storage file, and an account balance. Any user can create a contract by posting a transaction to the blockchain. The program code of a contract is fixed when the contract is created, and cannot be changed. A contract's storage file is stored on the public blockchain. A contract's program logic is executed by the network of miners who reach consensus on the outcome of the execution and update the blockchain accordingly. The contract's code is executed whenever it receives a message, either from a user or from another contract. A user can sends a message to a contract by including the message data and the address of the contract in her transaction. One contract can send a message to another using a special instruction in its program code. While executing its code, the contract may read from or write to its storage file.

[10] A contract can also receive money into its account balance, and send money to other contracts or users. Conceptually, one can think of a contract as a special “trusted third party” however, this party is trusted only for correctness and availability but not for privacy. In particular, a contract's entire state is visible to the public. A contract's code will be invoked whenever it receives a message. A contract can define multiple entry points of execution in Ethereum Serpent language, each entry point is defined as a function. A message contents will specify the entry point at which the contract's code will be invoked. Therefore, messages act like function calls in ordinary programming languages. After a contract finishes processing a message it receives, it can pass a return value back to the sender.

[11] Ethereum uses the concept of “gas" to discourage over - consumption of resources (e. g., a contract program that causes miners to loop forever). The user who creates a transaction must spend currency to purchase gas. During the

execution of a transaction, every program instruction consumes some amount of gas. If the gas runs out before the transaction reaches an ordinary stopping point, it is treated as an exception: the state is reverted as though the transaction had no effect, but the Ether used to purchase the gas is not refunded! When one contract sends a message to another, the sender can offer only a portion of its available gas to the recipient. If the recipient runs out of gas, control returns to the sender, who can use its remaining gas to handle the exception and tidy up.

What smart contracts are for?

With so many things that smart contracts cannot do, one might ask what they're actually for. But in order to answer this question, we need to go back to the fundamentals of blockchain themselves. To recap, a blockchain enables a database to be directly and safely shared by entities that do not trust each other, without requiring a central administrator.

[2] BlockChain enable data disintermediation, and this can lead to significant savings in complexity and cost. Any database is modified via "transactions", which contain a set of changes to that database which must succeed or fail as a whole. For example, in a financial ledger, a payment from Alice to Bob is represented by a transaction that (a) checks if Alice has sufficient funds, (b) deducts a quantity from Alice's account and (c) adds the same quantity to Bob's.

In a regular centralized database, these transactions are created by a single trusted authority. By contrast, in a blockchain - driven shared database, transactions can be created by any of that blockchain's users. And since these users do not fully trust each other, the database has to contain rules which restrict the transactions performed [2].

For example, in a peer - to - peer financial ledger, each transaction must preserve the total quantity of funds, otherwise participants could freely give themselves as much money as they liked.

One can imagine various ways of expressing these rules, but for now there are two dominant paradigms, inspired by bitcoin and Ethereum, respectively. The bitcoin method, which we might call "transaction constraints", evaluates each transaction in terms of: (a) the database entries deleted by that transaction and (b) the entries created.

In a financial ledger, the rule states that the total quantity of funds in the deleted entries has to match the total in those created. (We consider the modification of an existing entry to be equivalent to deleting that entry and creating a new one in its place).

[13] [11] The second paradigm, which comes from Ethereum, is smart contracts. This states that all modifications to a contract's data must be performed by its code. (In the context of traditional databases, we can think of this as an enforced stored procedure.) To modify a contract's data, blockchain users send requests to its code, which determines whether and how to fulfill those requests. As in this example, the smart contract for a financial ledger performs the same three tasks as the administrator of a centralized database: checking for sufficient funds, deducting from one account and adding to another.

Both of these paradigms are effective, and each has its advantages and disadvantages. To summarize, bitcoin - style transaction constraints provide superior concurrency and performance, while Ethereum - style smart contracts offer greater flexibility.

So to return to the question of what smart contracts are for: Smart contracts are for blockchain use cases which can't be implemented with transaction constraints.

Given this criterion for using smart contracts, I'm yet to see a strong use case for permissioned blockchain which qualifies. All the compelling blockchain applications I know can be implemented with bitcoin - style transactions, which can handle permissioning and general data storage, as well as asset creation, transfer, escrow, exchange and destruction. Whatever the answer turns out to be, the key to remember is that smart contracts are simply one method for restricting the transactions performed in a database. This is undoubtedly a useful thing, and is essential to making that database safe for sharing. But smart contracts cannot do anything else, and they certainly cannot escape the boundaries of the database in which they reside.

Game Theory in Blockchain Technology (Prisoner's Dilemma)

[1]Game theory motivates people to collaborate to protect their interests or gain a reward. For the uninitiated, game theory is a branch of mathematics that studies the strategic interaction among rational actors. [1]The prisoner's dilemma is an experiment analyzed in game theory that shows why a group of people might have a problem cooperating, even when it seems like they'd all be better off by cooperating.

In this thought experiment, there are two criminals who are brought in for questioning for their suspected participation in a crime. Both are in separate interview rooms, and let's assume they're called prisoner A and prisoner B. (hence, the name), and are potentially facing life imprisonment.

In the scenario, each can choose either to rat on the other person or to stay quiet.

Now, cops put on several conditions. If prisoner A and prisoner B both stay quiet, then they both walk free. This is clearly the best outcome. But if A talks while B stays quiet, then the cops will put B away for life, and let A off with only 3 years in jail. The same is true if B talks. Now the cops put another condition that if they both end up talking, they both go to jail for, say, 10 years. This is better than life but worse than going free. Say you're prisoner A. You don't know what is going on in B's mind. Logically, you will find out that it's better to talk. If both of you stay quiet, you get to leave. But, if B talks and you stay quiet: you go away for life. On the other hand, if you talk, the worst thing that can happen is that you get 10 years. Thus, in situations like this, the cops end up with everyone talking – even though they'd all be better off if no one talked. Because of lack of trust between the prisoners, they lost their freedom.

Industries moving towards digitization are finding themselves in a somewhat similar situation right now. Automation has had the greatest impact on how businesses operate – cutting their costs, increasing their productivity and speeding up their work. But despite this digital transformation, businesses have time and again struggled to automate the foundation of trade;

Trust

Every person on the planet, who has ever made a purchase, from buying something off Alibaba to purchasing a house, knows how complex it is for two parties to complete a transaction. Funds must be verified, disclosures must be made in writing, and asset transfer needs to be done, and so on. This is where businesses have a few questions that need to be answered.

• *Does this party actually and legally own the asset I want to buy?*
• *Will this party actually give me the amount they promised?*
• *How can I ensure that this party delivers the goods once I give them the money?*

Usually, these questions are answered by third parties. This is the biggest issue – the lack of trust between the participating parties, and forced reliance on third party. This is where I feel blockchain "Automated Trust" comes into play. It doesn't really matter how many offbeat use - cases the technology has, but the central, the core use - case of blockchain would always be automating trust. Blockchain can significantly reduce the operational friction, costs and headaches associated with business processes – hence is a key technology that companies associated with financial services, supply chain, energy, healthcare, retail, and automotive sectors are exploring.

Application of Game Theory in Smart Contracts

Game theory plays a key role in cryptocurrencies like bitcoin. Whether it's the payoff matrix, Nash equilibrium or the prisoner's dilemma, game theory concepts have big implications for blockchain and smart contracts.

Blockchain and smart contracts will grow the game theory field. Game theory is a major consideration when designing many blockchain and smart contracts applications. Blockchain businesses, cryptocurrencies and smart contract - based crypto - tokens need sound game theory or else they won't last. When miners spend time and energy on electricity and computing the hash of a block, then they need that block to be correct so it is accepted by the network. That's how they receive the block's mining reward. Otherwise miners lose time and money, and eventually everyone refuses to interact with their node. Bitcoin miners competing, proving they produced a specific block, as well as half the network having to agree on the next block, is the first game theory to work in the context of digital currencies. In bitcoin's decentralized model, if miners want to earn rewards, they have to abide by the rule of bitcoin.

Bitcoin's game theory, which is driven largely by the network's mining complex, inspired Ethereum. Although it uses the same game theory as Bitcoin, Ethereum empowers developers to design and implement their own game theory systems in the form of smart contracts.

A smart contract denotes a computer protocol that facilitates, verifies or enforces the negotiation or performance of a contract. I believe that smart contract systems have a long way to go before they are robust enough upon which to base business. And now, a new concept of oracles is gaining traction.

[14] Oracles employ their own unique types of game theory to facilitate the provision of factual data about the real world in a distributed system, and help blockchain reflect the real world with accuracy. An oracle is basically a game theory that forces people to report correctly that which has transpired in the real world or they get financially penalized. If it's done correctly, they get financially incentivized. Blockchain incentivizes people to tell the truth about the world.

Are smart contracts viable for large scale applications?

Finally I come to my aim of writing this paper; many companies have pitched many smart contract use cases, and have found themselves responding, time and again, that they simply cannot be done. From this result, I have identified the three smart contract misconceptions that are most commonly held. These ideas aren't wrong because the technology is immature, or the tools are not yet available. Rather, they misunderstand the fundamental properties of code which lives in a database and runs in a decentralized way.

1) Contacting External Services

This is one of the limitations, due to the deterministic nature of the blockchain, smart contracts are not allowed to directly access public data available on the Internet. For example, to get a simple exchange rate, you will need to make use of an external third party (oracle) [14]. This complicates the smart contract's code and adds another layer of complexity because external services' jobs need to be coordinated. Also, you will have to check that the oracle didn't forge the response.

Often, the first use case proposed is a smart contract that changes its behavior in response to some external event. For example, an agricultural insurance policy which pays out conditionally based on the quantity of rainfall in a given month. The imagined process goes something like this: The smart contract waits until the predetermined time, retrieves the weather report from an external service and behaves appropriately based on the data received. This all sounds simple enough, but it's also impossible. Why? Because a blockchain is a consensus - based system, meaning that it only works if every node reaches an identical state after processing every transaction and block. Everything that takes place on a blockchain must be completely deterministic, with no possible way for differences to creep in. The moment that two honest nodes disagree about the chain's state, the entire system becomes worthless. Smart contracts are executed independently by every node on a chain. Therefore, if a smart contract retrieves some information from an external source, this retrieval is performed repeatedly and separately by each node. But because this source is outside of the blockchain, there is no guarantee that every node will receive the same answer.

Perhaps the source will change its response in the time between requests from different nodes, or perhaps it will become temporarily unavailable. Either way, consensus is broken or the entire blockchain dies. So, what's the workaround? Actually, it's rather simple. Instead of a smart contract initiating the retrieval of external data, one or more trusted parties ("oracles") create a transaction which embeds that data in the chain. Every node will have an identical copy of this data, so it can be safely used in a smart contract computation. In other words, an oracle pushes the data onto the blockchain rather than a smart contract pulling it in.

When it comes to smart contracts causing events in the outside world, a similar problem appears. For example, many like the idea of a smart contract which calls a bank's API in order to transfer money. What if every node is independently executing the code in the chain, which is responsible for calling this API?If the answer is just one node, what happens if that particular node malfunctions, deliberately or not? And if the answer is every node, can we trust every node with that API's password? And do we really want the API called hundreds of times? Even worse, if the smart contract needs to know whether the API call was successful, we're right back to the problem of depending on external data. As before, a simple workaround is available. Instead of the smart contract calling an external API, we use a trusted service which monitors the blockchain's state and performs certain actions in response. For example, a bank could proactively watch a blockchain and perform money transfers which mirror the on - chain transactions. This presents no risk to the blockchain's consensus because the chain plays an entirely passive role.

[13] Looking at these two workarounds, we can make some observations. First, they both require a trusted entity to manage the interactions between the blockchain and the outside world. While this is technically possible, it undermines the goal of a decentralized system. Second, the mechanisms used in these workarounds are straightforward examples of reading and writing a database. An oracle which provides external information is simply writing that information into the chain. And a service which mirrors the blockchain's state in the real world is doing nothing more than reading from that chain. In other words, any interaction between a blockchain and the outside world is restricted to regular database operations.

2) Enforcing on - chain payments

Here's another proposal that we tend to hear a lot: using a smart contract to automate the payment of coupons for a so - called "smart bond". The idea is for the smart contract code to automatically initiate the payments at the appropriate times, avoiding manual processes and guaranteeing that the issuer cannot default. Of course, in order for this to work, the funds used to make the payments must live inside the blockchain as well; otherwise a smart contract could not possibly guarantee their payment. Recall that a blockchain is just a database, in this case a financial ledger containing the issued bond and some cash. So, when we talk about coupon payments, what we're actually talking about are database operations which take place automatically at an agreed time.

While this automation is technically feasible, it suffers from a financial difficulty. If the funds used for coupon payments are controlled by the bond's smart contract, then those payments can indeed be guaranteed. But this also means those funds cannot be used by the bond issuer for anything else. And if those funds aren't under the control of the smart contract, then there is no way in which payment can be guaranteed.

In other words, a smart bond is either pointless for the issuer, or pointless for the investor. From an investor's perspective, the whole point of a bond is its attractive rate of return, at the cost of some risk of default. And for the issuer, a bond's purpose is to raise funds for a productive but somewhat risky activity, such as building a new factory. There is no way for the bond issuer to make use of the funds raised, while simultaneously guaranteeing that the investor will be repaid. It should not come as a surprise that the connection between risk and return is not a problem that blockchain can solve.

3) Hiding confidential data

Here comes the third challenge, as I've written about previously, the biggest challenge in deploying blockchain is the radical transparency which they provide. For example, if 10 banks set up a blockchain together, and two conduct a bilateral transaction, this will be immediately visible to the other eight. While there are various strategies for mitigating this problem, none beat the simplicity and efficiency of a centralized database in which a trusted administrator has full control over who can see what. Some people think that smart contracts can solve this problem. They start with the fact that each smart contract contains its own miniature database, over which it has full control. All read and write operations on this database are mediated by the contract's code, making it impossible for one contract to read another's data directly. This tight coupling between data and code is called encapsulation, and is the foundation of the popular object - oriented programming paradigm.

So, if one smart contract can't access another's data, have we solved the problem of blockchain confidentiality? Does it make sense to talk of hiding information in a smart contract? Unfortunately, the answer is no and this is because even if one smart contract can't read another's data, that data is still stored on every single node in the chain. For each blockchain participant, it's in the memory or disk of a system which that participant completely controls. And there's nothing to stop them reading the information from their own system, if and when they choose to do so.

Hiding data in a smart contract is about as secure as hiding it in the HTML code of a web page. Sure, regular web users won't see it, because it's not displayed in their browser window. But all it takes is for a web browser to add a 'View Source' function (as they all have), and the information becomes universally visible.

Similarly, for data hidden in smart contracts, all it takes is for someone to modify their blockchain software to display the contract's full state, and all semblance of secrecy is lost.

2. Conclusion

Smart contracts hold tremendous power, but they do have limitations. It is important to note that these systems are only as good as the people building them. So far, many smart contract systems have failed due to unforeseen bugs and events that were not part of the initial design. In many cases, these were merely technical flaws that can at least be fixed in time. However, with the recent rush to use blockchain technology for everything, we are likely to start seeing more substantial failures as people fail to understand the limits of the technology. For blockchain to truly have maximum business impact, both its advantages and limitations have to be addressed.

Businesses have to keep in mind that blockchain is not a cure - all. There are very specific use - cases where blockchain could help businesses. Companies need to figure out which places actually need trust to be automated, and then proceed with them instead of blindly replacing their Databases with blockchain. The companies who provide blockchain development also need to understand the same. Companies that merely replace centralized DB with blockchain without understanding if the technology is actually 'automating trust' are harming the industry in general. Companies like Ethereum need to invest more in educating their clients, rather than spending purely on development. As the coming developer of blockchain platforms, I always say *"It's not that people don't understand what they want smart contracts to do. Rather, it's that so many of these ideas are simply impossible".*

References

[1] Bitcoin: A Peer - to - Peer Electronic Cash System. Nakamoto S.
[2] SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. Bonneau J, Miller A, Clark J, Narayanan A, Kroll JA, Felten EW. S&P.
[3] Bitcoin's Academic Pedigree. Narayanan A, Clark J. ACM Queue.
[4] Smart Contracts Make Bitcoin Mining Pools Vulnerable. Velner Y, Teutsch J, Luu L. FC.
[5] Mixing Coins of Different Quality: A Game - Theoretic Approach. Abramova S, Schöttle P, Böhme R. FC.
[6] Decentralized Prediction Market without Arbiters. Bentov I, Mizrahi A, Rosenfeld M. FC
[7] "Zeus": Analyzing Safety of Smart Contracts. Kalra S, Goel S, Dhawan M, Sharma S. NDSS.
[8] "Ethereum": A next - generation smart contract and decentralized application platform. VitalikButerin.
[9] Ethereum: A secure decentralisedgeneralised transaction ledger. Wood G.
[10] Fair Two - Party Computations via Bitcoin Deposits. Andrychowicz M, Dziembowski S, Malinowski D, Mazurek.
[11] Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab. Delmolino K, Arnett M, Kosba A, Miller A, Shi.
[12] eth. How do Ethereum mining nodes maintain a time consistent with the network? Ethereum Wiki, June 2016. https: //github. com/ethereum/wiki/wiki/Light - clientprotocol, Accessed on 12/6/2019.
[13] Ethereum. Light client protocol. Ethereum Wiki, May 2016. https: //github. com/ethereum/wiki/wiki/Light - client - protocol.
[14] Ethereum. The mix ethereumdapp development tool. GitHub, 2016. https: //github. com/ethereum/mix, Accessed on 20/5/2019.
[15] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In International Conference on Financial Cryptography and Data Security, pages 436–454. Springer, 2014
[16] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 30–41. ACM, 2014.26. L. Luu, D. - H. Chu, H. Olickel, P. Saxena, and A. Hobor.
[17] Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 254–269. ACM, 2016.27. P. McCorry, S. F. Shahandashti, D. Clarke, and F. Hao. Authenticated key exchange over bitcoin.
[18] In Security Standardization Research, pages 3–20. Springer, 2015.