2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)

The Scalability Challenge of Ethereum: An Initial Quantitative Analysis

Mirko Bez, Giacomo Fornari, Tullio Vardanega
Department of Mathematics
University of Padova, Italy
e-mail: {bez.mirko, fornarigiacomo}@gmail.com, tullio.vardanega@unipd.it

***Abstract*—The concerns held on the scalability of permissionless Blockchain platforms are a significant hindrance to their wider adoption. To address this issue rigorously, we consider it opportune to assess the current implementation and the associated improvement proposals, within a single coherent evaluation framework. This work addresses this specific question in the particular context of Ethereum, a prominent implementation of Blockchain, using a threefold approach. First, it maps the internal constituents of Ethereum onto a layered architecture inspired in the ISO/OSI model, so that its provisioning organization can be better understood. Second, it employs the AKF Scale Cube to analyze the pros and cons of the present specification of Ethereum, as well as of the current improvement proposals, so that the scalability challenges can be reasoned about in an orderly fashion. Finally, it uses an extensible test environment with synthetic benchmarks so that the transaction throughput of the current implementation of Ethereum can be evaluated in a private scenario, when no smart contract is to run. Our conclusions suggest that Ethereum respects the scalability trilemma being versed on security and decentralization renouncing scalability. These limits can be mitigated by adopting novel solutions such as Plasma and Sharding which enable a significant increment of performance by partitioning the data, and, ultimately, unlocking parallel execution of the transactions.**

***Keywords*-Ethereum, Scalability, CaaS, Blockchain**

I. Introduction

Ethereum [1] is a well-known incarnation of the Blockchain technology[1]. Whereas the Blockchain in its current form first appeared in 2008 [2], this technology has become a very active research field in ICT only very recently.

The aim of the Blockchain technology is to provide a total order of the transactions in a distributed ledger without relying on a trusted third party [3]. This objective must confront hard challenges like transaction repudiation and the infamous *double-spending problem* [2]. The former problem is self-explanatory; one solution to it uses digital signatures. The latter stems from using one digital token to pay multiple entities, altering the transaction history after a payment.

Whereas firsthand evidence of the practicality of decentralized solutions emerged with the rise of Bitcoin [4][2], Ethereum can be regarded as a generalization of it. The execution environment of the former (i.e., Script, the Bitcoin scripting system) is stateless and is used only to express preconditions on spending money, e.g., to demonstrate the possession of a given private key. In Ethereum, instead, the execution environment (i.e. the EVM, the Ethereum Virtual Machine) is stateful and Turing Complete. To avoid misuse of network resources, each opcode is associated with a finite amount of gas: that limit alone guarantees that all computations always eventually terminate. One typifying example of application of the Blockchain is the supply chain. In this scenario, the blockchain serves as a trustless ledger in which both consumer and producer can keep track of the different stages in the life cycle of sensitive products, especially food products, which are more sensitive to environmental influences. The distributed ledger may be used also to produce transnational certificates, which can be verified independently of the parties involved, without a central authority to guarantee its validity. When put in the context of emerging scenarios like the Internet-of-Things, the Blockchain can be understood as a specialized self-contained service that helps address these and other challenges. In this respect, the Blockchain can be seen as a Computation-as-a-Service platform apt to provision distributed computational resources with high fault tolerance guarantees.

However, in spite of its attractive potential, the mass adoption of this technology is prevented by its inherent limitation in the scalability. The most patent concern with respect to scalability for all permissionless cryptocurrencies in general is their limited *maximum transaction throughput* [5], [6]. The inherent characteristic of the Blockchain is that the time to process individual transactions grows linearly with the increase of participants. As the choice of the transactions included in the Blockchain depends on the free – and presumably selfish – will of miners, it is more likely that only the more profitable transactions (the ones that earn greater fees for the miner) would be included in the distributed ledger, to the detriment of the less profitable ones [1].

The *bootstrap time*, i.e., the time needed for a new node to download and process an entire transaction history, is another key factor of influence on scalability (toward *scaling out*) [6], and the ability of new nodes to join the network.

In addition to these factors, a number of confirmation blocks are needed, to assure that the transactions are really confirmed [5], i.e., a block followed by a number of other

[1]In this paper we use the term Blockchain (with capital B) to refer to the Blockchain technology, while the term blockchain (with lower b) to denote the data structure that stores the public ledger in each node.

[2]Please refer to [4] for a survey on the history and ancestors of Bitcoin.

978-1-7281-1442-2/19/$31.00 ©2019 IEEE
DOI 10.1109/SOSE.2019.00031

blocks is unlikely to be discarded. This uncertainty in the acceptance of a block is also known as *lack of Consensus Finality* [3], which is due to the probabilistic nature of the Proof-of-Work (PoW) consensus algorithm.

In Bitcoin, the number of recommended confirmation blocks currently is 6, whereas in Ethereum it is around 12. For Bitcoin, this requirement means that, upon the insertion of a transaction in a block, at least an hour must pass to be sure that it has reached a high enough degree of immutability. In Ethereum, this interval shrinks to 3 minutes[3].

Another concern with this technology is the need to store an increasingly large amount of data to assure sufficient security guarantees. At the time of this writing, the minimum requirement to run a Bitcoin full-node, that is, a node that verifies all transactions, is 200 GB[4].

In the particular case of Ethereum, the introduction of stateful smart contracts makes the problem similar to database replication, notably to *state machine replication* [3]. Wood [1] argues that it is intrinsically very hard to reach a high degree of scalability by parallelizing transactions in this system because it is essentially a state transaction machine. Indeed, the state in Ethereum influences the smart contract execution and therefore the majority of transactions are dependent on previous ones, which makes Ethereum *stateful*, and therefore ill fit for scalability. Also Buterin in his (now deprecated) Mauve Paper [7] argued that Ethereum cannot be more efficient than a single machine. On these accounts, we regard the problem of the scalability of Ethereum as fully worth of investigation. In this paper, we explore ways to address the intrinsic limitations of this platform, and provide a qualitative assessment of how far certain solutions can improve its transaction throughput.

The contributions we make in this work are threefold:

- First, we propose an interpretation of the architectural organization of Ethereum by a stack of specialized service layers, similar in concept to the ISO/OSI model. This mapping helps reason in a structured manner on the provisioning architecture of Ethereum.
- Second, we use the AKF Scale Cube [8] – providing a rationale for this choice – to assess systematically how Ethereum fares with respect to the multiple dimensions of scalability, and how the current and future proposals might improve it.
- Third and last, we present an extendable tool to apply quantitative metrics for the performance analysis of Ethereum implementations, and use the results of its use to evaluate the maximum transaction throughput of the current implementation, trialed in a private scenario where no smart contract is executed.

The remainder of this paper is organized as follows. Section II presents related work of consequence to our investigation. Section III describes our interpretation of the architecture of the Ethereum platform. Section IV investigates the scalability of Ethereum and the improvement proposals with the aid of the Scale Cube [8] and relates the notions behind it to other works in the state of the art. Section V presents our test and the obtained results. Finally, Section VI draws some conclusions and outlines future work.

II. Related Work

The works [9] and [10] illustrate two different layered views of permissionless Blockchain systems. The former distinguishes the Consensus and the Network layer. The latter singles out the application, the Execution Engine (shorted to EVM in this paper), the Data model, and the Consensus Layer. Our own interpretation can be regarded as a specialization of the general model described in [9] in which the two bottom layers correspond to the Network Layer and the upper four layers correspond to the Consensus layer. The reason why our model differs from the one in [10] is made apparent in Section III, which highlights the role of the EVM to serve the Data, Consensus and Application layers.

The idea of using the Scale Cube [8] to categorize and evaluate qualitatively how Ethereum fares for scalability, and how do the most prominent improvement proposals, resembles what has been done for the family of Blockchain consensus protocols in [3] and the taxonomy provided in [5], which helps application architects to select the best-suited Blockchain solution. In particular, we analyze the improvement proposals Proof-of-Stake (PoS) [11], Plasma [12] and Sharding [13]. The Ethereum foundation [7], [13] and the scientific community [6] have proposed sharding combined with PoS (*shasper*[5]) as an effective measure to dramatically increase the transaction throughput. Although the details about this proposal are being constantly updated[6] and no reference implementation of it exists yet[7], we discuss here the basic ideas [7], [13] about sharding, and explain why this approach could improve the scalability of the platform.

We acknowledge that when assessing qualitatively the scalability of a system, the Scale Cube is not the only solution [14]. Other viable approaches might examine which "Scalability Design Patterns" [15] or which characteristics of scalable systems [14] does Ethereum exhibit. However, we preferred using the Scale Cube because we deem it to map the principal qualitative characteristics of scalability onto a simple yet expressive conceptual model, which addresses: functional decomposition; horizontal replication; and horizontal data partitioning.

[3]This estimate assumes that in Bitcoin a new block is generated every 10 minutes, while in Ethereum every 15 seconds.

[4]https://bitcoin.org/en/bitcoin-core/features/requirements

[5]This name is obtained by the contraction of the words *sharding* and *Casper*, the Ethereum's PoS proposal.

[6]https://github.com/ethereum/eth2.0-specs

[7]The Prysmatic Labs company is modifying the geth implementation to augment it with both PoS and sharding (https://github.com/prysmaticlabs/prysm), but its current status is far from production ready.

Numerous works attempted to assess quantitatively the scalability of permissionless Blockchains (in particular Bitcoin) systems. Screening their rationale, we recognize three main approaches to assess quantitatively the scalability or the security of a Blockchain system, as follows:

1) To leverage the existing live network, as done for example in [16], where a modified client was attached to the Bitcoin network to assess the influence of block size on the propagation time.
2) To adopt a mathematical model, as done in [9] to evaluate the security and scalability of a PoW system with different parameters.
3) To create a synthetic test environment and develop benchmarks to mimic the working of system variant.

Option 1) allows checking the status of a long-lived system with a large number of geographically distributed nodes, which cannot be reproduced easily in a lab environment. This solution, however, does not provide sufficient flexibility to investigate the influence of some parameters on scalability.

Option 2) was not viable, for we lacked an in-house simulation tool for Ethereum; even if we had it, it would be of little use unless it was validated against a real-world reference, which would be hard to achieve.

Hence, we focused on option 3), which allowed us to tune the experiment parameters at will and availed us a controlled environment. Other followers of this approach include the evaluation of the Bitcoin-NG [17], and the Algorand proposal [18], which deploy a large test network with thousands and tens of thousands of nodes, respectively. For our study, we did not have as many resources. We therefore took inspiration from Blockbench [10], in which Hyperledger Fabric[8], Parity (a network in which all nodes use the parity client instrumented with Proof of Authority) and Ethereum (geth[9] with the usual PoW consensus algorithm) are compared in a permissioned scenario, with authenticated nodes.

Regarding [10], however, we contend that the test parameters that influence the transaction throughput are not sufficiently justified: for example, the difficulty is manually adjusted[10] and there is no description for the selected gas limit and the initial timestamp, which have an influence on the test difficulty. This lack of justification cannot be balanced during the test because of the short test time (10 minutes).

In Section V, we provide a justification for our choice of parameters. In particular, we propose an initial test to find a reasonable initial value for the difficulty parameter.

III. Understanding the Ethereum Architecture

Although Ethereum clients with different implementations can agnostically participate in the system, there is no common agreement on the Ethereum architecture. Since "an implementation is not an architecture" [8], we propose our own conceptual model, inspired in the OSI reference model, and distilled from different sources and architecture proposals. Fig. 1 depicts the model that we have inferred. It is comprised of 5 stacked layers, respectively, the network, propagation, data, consensus and application layers, and one additional layer, constituted by the Ethereum Virtual Machine (EVM), which we place vertically because it transversely serves the data, consensus, and application layers. The EVM is vertical in our architectural model because, when a smart contract – which belongs to the application layer – is executed, the data contained in the data layer is deterministically transformed by the EVM. The consensus layer verifies the results of any such computation, in order to guarantee data consistency.

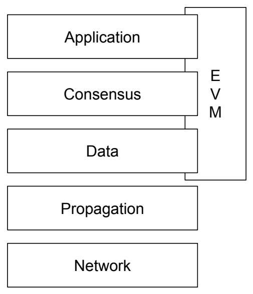

Figure 1. The layers of the Ethereum architecture in our reckoning.

The aim of the network layer is to build the peer-to-peer network, a decentralized architecture in which all nodes are logically equivalent and operate as *servants*, that is, nodes that act as both client and server at the same time. In this type of architecture, the nodes are formed by processes and the links represent the possible communication channels, which form a *structured overlay network* across which the nodes can propagate information efficiently. Effectively, this layer is constituted by the RLPx Node Discovery Protocol [19], which is a slight modification of the Kademlia protocol [20].

The propagation layer leverages the contact information obtained from the network layer to spread the blockchain information among all peers. In the propagation layer, we can identify three main components:

- Recursive Length Prefix (RLP) algorithm [1], the main encoding method used to serialize objects in Ethereum;
- the *RLPx Transport Protocol*, which plays a very similar role to the OSI transport layer intended[11];
- the *Ethereum Wire Protocol*, which propagates the blockchain information.

The Data layer consists of the data structures and the data types that are present in the Ethereum system. With the term *data types*, we refer to the abstract data types that identify the common Ethereum objects, such as the state,

[8]https://www.hyperledger.org/projects/fabric
[9]An Ethereum client, at https://github.com/ethereum/go-ethereum.
[10]From the public repository, it is clear that the granularity used for the adjustment is very coarse, and thus too imprecise.
[11]Cf. https://github.com/ethereum/devp2p/blob/master/rlpx.md#transport

the accounts, and the transactions. The state is the mapping between account addresses and account states. The accounts are of type contract and Externally Owned Account (EOA). They are essential for the user to interact with the Ethereum blockchain via transactions. The transaction is a single instruction constructed by an actor external to Ethereum [1], and it is serialized and included in the blockchain. It can be used to either transfer Ether or trigger contract code execution. A transaction represents a message call or the creation of a new account with associated EVM code.

The consensus layer defines how the nodes reach consensus on the state held in the data layer, which involves the mining and the transactions execution process.

When we contemplate a PoW system like in Ethereum, we recognize three basic rules that each node should follow to preserve data consistency:

1) To accept only valid blocks according to given validation criteria.
2) To create new valid blocks through a PoW algorithm, e.g., the latest version of the Dagger-Hashimoto algorithm for Ethereum [1].
3) To use a selection rule to choose between two different valid forks depending on the amount of work performed in each fork. (In Bitcoin and Ethereum, said rule corresponds to the longest chain rule, although some sources in the Ethereum Community suggest that they implemented an easier version of the GHOST protocol [9].)

The application layer comprises the smart contract, a collection of functions enabled to modify the state. These are executed on the EVM, which can be regarded as the business logic of the system. Usually, a smart contract is written in a high-level language, such as Solidity or Vyper, subsequently compiled to EVM bytecode. The contracts communicate with one another and with external actors through their Application Binary Interface (ABI). This interface serves as a declaration of the functions implemented by the contract and their arguments, through which another actor can trigger the execution of it and get results back.

One aid for the Blockchain to be regarded and employed as a self-contained service in a larger Service-Oriented architecture comes from the JSON RPC API [21], which allows external actors to invoke the exposed API methods by sending JSON encoded requests.

IV. Scalability Analysis

A. Motivation

Ethereum is presented as a "technology on which all transaction-based state machine concepts may be built" [1]. This claim suggests that, even systems exposed to high volumes of transactions, might be built with Ethereum.

From the scalability perspective, Blockchain-based systems like Ethereum and Bitcoin, are commonly compared (cf. [22] and [6]) with conventional payment systems like VISA. The latter is capable of processing more than 2000 transactions per second, with a peek at 56,000, versus the disappointingly-low 7 of Bitcoin [6] and the 15 of Ethereum, as confirmed by the tests we discuss in Section V[12]. This quantitative comparison shows that Ethereum is quite far from offering a viable implementation platform for all transaction-based systems, owing to evident limitations in scalability. In fact, the lack of scalability is the practical issue that hinders the mass-adoption of Blockchain-based systems, in particular Ethereum. Unsurprisingly, the Ethereum Foundation announced two grant programs intended to fund research and development projects specifically focused on solving the scalability challenge[13].

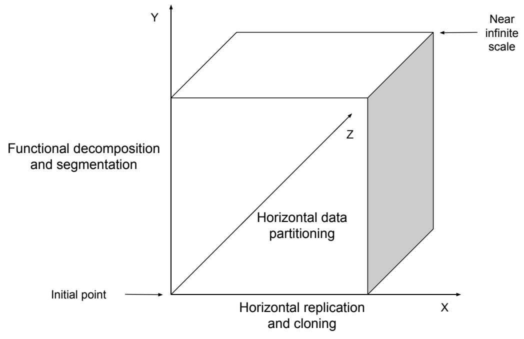
Y
Near infinite scale
Z
Functional decomposition and segmentation
Horizontal data partitioning
Initial point
Horizontal replication and cloning
X

Figure 2. The Scale Cube.

B. The Scale Cube

We discuss the Ethereum scalability along the three axis of the Scale Cube [8], which we regard as an easy-to-understand conceptual model that helps evaluate qualitatively how scalable the architecture of a system is. Using this model, we can appreciate the weaknesses of the system and the improvement proposals systematically and uniformly. This approach may provide a simple and fair basis to analyze future works and future scalability proposals.

The *Scale Cube*, shown in Fig. 2, uses the representation of a cube drawn on a 3-dimensional space to define the three different scaling directions that an architecture must address to grow and shrink along with demand. The Scale Cube does not provide actual metrics to quantify scalability in any dimension, but rather a way of thinking about scaling; that is what we mean with the *scaling directions* tag in the picture.

The use of one or two axes does not preclude the possibility to scale on the third axis. The initial point with coordinates (0, 0, 0) means the least scalability. The prototypical system at the initial point consists of a single monolithic application and

[12]This value does not deviate too much from the Ethereum Transaction History of Etherscan (see https://etherscan.io/chart/tx).

[13]https://blog.ethereum.org/2018/01/02/ethereum-scalability-research-development-subsidy-programs/

storage retrieval system likely running on a single physical machine [8]. Of course, it might scale up, that is, it could run on a more powerful machine, but it won't scale out, hence it will not draw any benefit from a distributed architecture. All of the three axes scale well from a transaction perspective, that is, in our case, transaction throughput. In Sections IV-C, IV-D and IV-E, we discuss where Ethereum places with respect to each individual axis, and we examine solutions proposed to increase the scalability of the system.

C. X-Axis: Horizontal Replication

The x-axis of the Scale Cube is concerned with horizontal replication and cloning of services and data with absolutely no bias, running each one and identical copy of the system on a different server. Usually, the work is distributed by a load balancer. Reasoning on the x-axis is easy and the implementation can be straightforward, but the persistent data sets on which each clone works have to be replicated in their entirety, which much increases operational costs.

Ethereum's current status and improvement proposals: The consensus algorithm of Ethereum is adaptive to ensure that the security of the network is not impacted by the work rate of miners: increasing the global hash rate by adding miners or speeding up their work, causes the creation of blocks to become more difficult.

In Section V-A, we measure the maximal transaction throughput in a private Blockchain[14]. Our tests show that incrementing the number of miners in the network does not increase the transaction throughput, once the difficulty stabilizes. We notice that the results between different runs may vary, probably because of the short run time of our tests, which reduces the opportunity to reach a stable point. The turbulence in performance can be due to a variety of factors like network overhead, the miners' influence on the mining difficulty, or perturbations in the performance of the nodes. Another significant test in this regard is described in Section V-A: in it, we set a gas limit high enough to not restrict the number of transactions that can be included in a block. The results show a little increment in the transaction throughput with regard to the previous test, but again the number of miners does not affect the overall performance. We can therefore conclude that replicating the miners in the network does *not* increment the transaction throughput, dissolving any scaling ability on the x-axis in the current implementation. Mind you, our notion of replication here does not require all nodes to run the same client. Indeed, there are multiple client implementations (e.g., geth, parity, etc.), which all conform to the same protocols, and therefore are functionally equivalent.

In Section I, we already noted that Ethereum is stateful, which makes it intrinsically ill fit for scaling on the x-axis. Moreover, the implementation of the consensus layer requires that every node validates each block propagated in the network, executing all the transactions and the EVM computations, which is in striking contrast with the notion of load distribution evoked in the x-axis. Hence, the performance of the single computer replicated in the network determines the maximum performance of the system overall. Because of these issues, it is very difficult to identify radical ways to scale on the x-axis without losing other fundamental characteristics, like decentralization and the lack of need for trusting other parties. Without the latter features, we would have a traditional system with a central authority on which we could implement traditional scaling solutions.

[14]See Section V for a detailed tests description.

D. Y-Axis: Functional Decomposition

The y-axis scaling focuses on the responsibility-based separation of staged actions, which can be determined by the type of data or the type of work performed for a transaction. While scaling on the x-axis means in effect "everyone does everything", scaling on the y-axis specializes the workers by splitting the whole service in small unitary services, assigning each to a single worker. If we combine the x-axis with the y-axis, we could have a cluster of specialized workers, in which each cluster covers a different service (stage), but workers in the same cluster do the same job. With regard to data persistence, unlike what happens in the x-axis, each service should have its own non-shared data, thus segmenting (aka sharding) them based on what each individual service needs to access. This solution implies that the resource demand for each service can be sized and optimized so as to reduce operational costs.

Ethereum's current status and improvement proposals: At present, Ethereum simply does not scale on this axis. Separation of responsibilities based on the type of work might be achieved per node type. A miner is responsible for collecting pending transactions and building a valid block, while every node is responsible for verifying the blocks generated by the miners. Yet, as already noted in Section IV-C, the transactions throughput does not increase with the cardinality of miners. Among the improvement proposals from the Ethereum community, no single one appears to address this direction. In the Proof of Stake (PoS) proposal, we can single out different node functions as we do in the PoW.

PoS is a class of algorithms through which a cryptocurrency Blockchain network achieves distributed consensus. While with PoW, truth is established by heavy computation performed by the miners, the PoS has validators instead of miners, and the consensus depends on the *stake* confirmed by each validator, which consists in an economic deposit in the network's cryptocurrency (Ether in this case).

Fig. 3 depicts the creation of a new block, in concept. If a node owns an amount of the Blockchain's base cryptocurrency, it can become a validator by sending a deposit transaction, which locks a given value. Once the

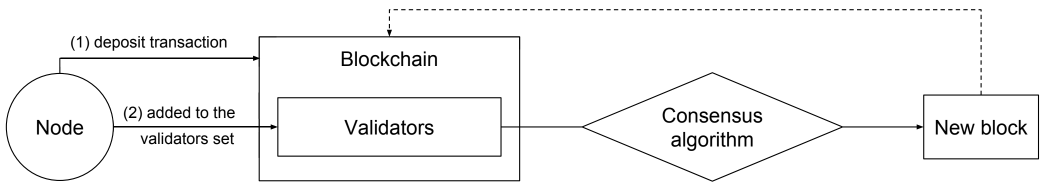

Figure 3. Overview of a block creation in the Proof of Stake.

transaction has successfully executed, the node becomes one of the validators. The consensus algorithm determines how a validator is chosen in the set of validators to *propose* the next block, and then how the validators agree on which block is canonical. Different types of PoS can be obtained by varying the consensus algorithm and how the rewards are assigned. The different node roles that can be identified in this solution do not support any kind of scaling on the y-axis. Indeed, although implementing different functions, those roles do not define a proper functional decomposition since they all operate on global shared data.

E. Z-Axis: Horizontal Data Partitioning

The obvious route for scaling on the y-axis is to segment the system based on the (type of) service, that is, after *dissimilar* things. Conversely, scaling on the z-axis means segmenting on *similar* things, thus slanting segmentation after the data or the actions that are unique to the sender or the receiver of the request.

In the particular case of the Ethereum's inner composition, to enhance significantly the system, a z-axis split should partition both transactions and the data necessary to perform them. A typifying example of such scaling in a client-server architecture could be the geographic distribution of the servers based on the clients requests, such that a request originated form a region is processed by a server located in that region instead of anywhere else.

Ethereum's current status and improvement proposals: Current Ethereum is not well versed in the z-axis, owing to the very nature of the blockchain: to process transactions, each network node should have information about the whole blockchain as well as the status of all network accounts.

Furthermore, a z-axis split similar in nature to the one of MongoDB [23], i.e. letting different nodes store different parts of the state, would surely decrease the amount of data to store on each node, but also it would:

A) require a lot of data to be sent across the network to perform the computations, and
B) fail to increase the transaction throughput, but rather eventually decrease it.

Conclusion A) follows from the observation that many computations require the ability to access a fair amount of addresses and their storage as the following example clarifies.

Consider the process of a transaction that requires some computations on the EVM. To complete this action we need to acquire some specific data items, such as the balance of the sender of the transaction and the code of the contract invoked. In turn, the contract being invoked may call other contracts and so on transitively. In addition, any of the execution actions down the line may modify the balance of other accounts as a side effect, e.g. through the `SELFDESTRUCT`[15] opcode. Thus, to process a transaction we need the account states of the sender and the called contracts and all account states that are affected by the computations. Claim A) implies claim B) in that the communication overhead incurred in acquiring all the required data items would likely slow down the transaction processing action. With these considerations in mind and by recalling that, to be significant, a z-axis split should partition both the transactions and the data necessary to perform the transactions, more clever approaches were proposed by the Ethereum community.

Among the proposals that move toward a z-axis split, the most prominent ones are Plasma [12] and Sharding [7], [13]. The Plasma proposal is categorized as an *off-chain* solution in that it executes some transactions outside of the main chain, effectively offloading them (more on this in Section IV-E1). Conversely, the Sharding proposal is categorized *on-chain*, since every transaction is executed inside the Ethereum network, without however requiring that every node should process every transaction, but rather the ones in its own shard (cf. Section IV-E2). Typically, in order to apply an off-chain solution, one or more smart contracts are sufficient (for example, Buterin proposed a specification for a minimal implementation called Minimum Viable Plasma[16]), while an on-chain solution requires a hard-fork, because it requires a radical change in the Ethereum protocol.

1) Plasma: The idea of Plasma is to create a tree hierarchy of blockchains. Each chain refers to a parent chain, except the root (i.e. the main chain, Ethereum in our case). Plasma consists of a series of smart contracts, which allows for many blockchains within a root blockchain [12]. The Plasma blockchains co-exist with their own business logic and all the computations are enforced at the root level only in the event of proof of fraud, and only the block

[15]This opcode causes deletion of the executing contract from the world state and sends the balance of the contract to a given destination account [1].

[16]https://ethresear.ch/t/minimal-viable-plasma/426

header hashes are submitted. During non-faulty states, only merkleized commitments are periodically broadcast to the root blockchain. The efficiency of this solution mainly owes to this feature since multiple state updates can be condensed in a single state update in the root chain. This system design implies that most of the computation can be done off-chain (i.e. outside of the root chain) and the state is enforced on-chain (i.e. in the root chain). Significant scalability for the users is achieved through chain split: when a Plasma blockchain grows too large, it can be split in child chains allowing the users to observe only Plasma blockchains in which their funds reside.

2) Sharding: The Ethereum sharding proposal consists in splitting the world state and transaction history into different partitions called shards. Each shard is a distinct universe, i.e. it is itself a PoS chain, in which the transactions affect only the accounts in the same shard. The transactions affecting one shard are collected in so-called *collations* by members of the network known as *proposers*. Collations in shards are the analogous of blocks in conventional blockchains, and, like blocks, they are chained and contain several fields, among which the list of transactions and the address of the proposer. Once per *epoch*, a block in the main PoS chain (main block) is created, where *cross-links* (e.g. the hash of the collations) to the accepted shard collations are inserted. The collations are accepted or refused by a committee of *attesters*. To become an attester or a proposer the members of the network should deposit an amount of Ether. After this operation, they can be randomly assigned to different shards. This is done to grant a certain level of decentralization and security.

This sharding proposal allows processing the transactions of different shards in parallel, which may increase the transaction throughput significantly. Another major benefit of this approach is that the nodes in one shard should verify only the transactions in their shard rather than all transactions. This simplified description of sharding does not contemplate *cross-sharding* communication mechanisms [13], which instead happens to be a desirable characteristic, which allows abstracting from data partitioning. Another planned feature is the transparency of sharding to smart contract developers [13]. Finally, another notable characteristic of this approach is that it requires the introduction of nodes that process the main chain and every shard. These nodes are named super-full nodes [13]. This notion leads to a major drawback similar to the one represented by the Application-Specific Integrated Circuits (ASIC) in the Bitcoin Proof-of-Work [1]. Indeed, the requirement of super nodes decreases the degree of decentralization by delegating to them the validation of the world state.

V. Evaluation Experiments

We designed and executed some tests to measure the maximal throughput for different configurations. As noted earlier, to analyze the scalability of a permissionless Blockchain system such as Ethereum, one should either rely on simulation [9] or run tests using thousands of nodes [17], [18]. We had neither a simulation of an Ethereum system nor as many computational resources at our disposal to equate a real implementation. To compensate for that shortcoming, we took inspiration from Blockbench [10], which compares the performance and scalability of Hyperledger Fabric and Ethereum in a *private* scenario, that is, considering a limited number of authenticated nodes.

Our first idea was to use the publicly available Blockbench repository[17]. Unfortunately, that route failed, due to an exceedingly large number of hard-coded configuration variables, and the lack of sound, clear and up-to-date documentation. Thus, we wrote our own test and benchmark suite.

A. Configuration

To keep the configuration easy, we opted for a classic master-slave logic. The master, i.e. the initiator and coordinator of the tests, uses the ssh protocol to run commands on the remote machines. Similarly to [10], we distinguish between the *miner* and *client* roles. The nodes of the former type are accountable to generate new blocks while the nodes of the latter type create and propagate transactions, and both verify the blocks[18]. We can assign multiple roles to a single machine. In this case, we run *one distinct geth instance* for each different role. The coordinator copies the right genesis file in the test machines. In each run of test, we distinguish two main phases: the setup and the test itself.

Setup phase: The setup consists in iterating on the test machines twice: firstly, the required ethash data structures are generated and secondly, the genesis file is used to create the genesis block and initialize the Ethereum World State. During the setup phase, the miners generate the Directed Acyclic graph (DAG) and the cache for the first two epochs, while the clients generate only the caches because they are required only to validate blocks. We generate the DAGs for the first two epochs, because ethash uses double buffer of DAGs to grant a smooth switch between epochs.

The genesis file is a JSON document that contains the information needed to create the genesis block and the initial state deterministically. The file contents include the network id, the timestamp, the initial difficulty, and the gas limit[19].

Obviously, each node of the network should be initialized with the same genesis file, otherwise the hash of the genesis block would differ and the peers would be unable to establish a connection with the Ethereum Wire Protocol [24]. Accordingly, the genesis file for the main network and the official Ethereum test networks are hard-coded. To create a new private Ethereum network it is sufficient to use a

[17]https://github.com/ooibc88/blockbench

[18]To reduce the number of test variables we consider only full nodes.

[19]A simple example is reported at https://github.com/ethereum/go-ethereum/wiki/Private-network

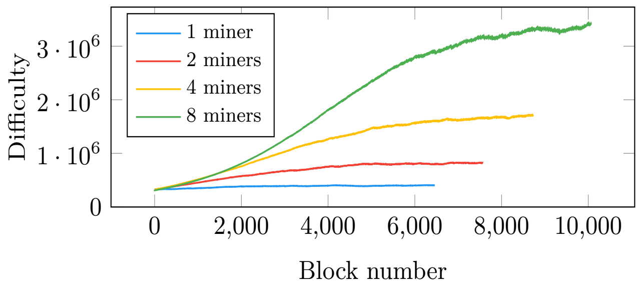

Figure 4. The growth of the difficulty in the 24 hour run.

new genesis file, with appropriate parameter changes. In the genesis file we employed for our experiments, we used an arbitrary network id and nonce, so that packets of different networks are simply dropped.

The main parameters with effect on the transaction throughput are the timestamp, the difficulty and the gas limit. In the next few paragraphs, we provide an in-depth description of these parameters and a justification for the values we used in the tests. As we had already observed, the values used in [10] for these parameters are not justified and the test duration (set at 10 minutes) is too short to reach stable points in all tests.

The *timestamp* of the genesis file, which corresponds to the one of the genesis blocks, affects the difficulty of the first blocks, and transitively of all blocks. A too old timestamp would cause a sudden and unwanted drop in the difficulty, which would nullify the value of short tests. To prevent these unwanted changes in the difficulty threshold, during the second loop of the setup phase, the initiator reads its timestamp and gives it as a parameter to the peers[20].

The *difficulty* is an adaptive parameter that determines how much effort should be invested in the creation of a new block. In our tests we used the same hardware and same operating system in all nodes and for all miners (we deliberately used only one thread for mining). Therefore, to find a suitable starting value for the difficulty for the different configurations, we ran a simulation with one, two, four and eight miners for 24 hours. Fig. 4 shows how the difficulty changed with the different number of miners. The plots in the figure suggest that, in our homogeneous system, the final values of difficulties have a quasi-linear dependency with the number of miners. For the initial value in our tests we took the median of the last 100 blocks. Table I reports these values, including the coefficient of variation, which shows that the values of the last 100 blocks are fairly stable indeed.

The *gas limit* in the genesis block sets the gas limit of the first block. The gas limit of the following blocks can be determined freely by the miners, as long as it always falls in the range determined by summing and subtracting to the gas limit a portion of itself [1]. The choice of the initial value thus is very critical for the soundness of short simulations.

[20]Before starting the test, all peers should be configured; using the coordinator's timestamp does not assume fine-grained coordinated clocks.

Table I
THE MEDIAN OF THE LAST 100 BLOCKS, VALUE USED IN THE TESTS AS START DIFFICULTY, AND THE COEFFICIENT OF VARIATIONS.

Number of Miners	Difficulty	Coefficient of variation
1	404559	0.0073%
2	821994	0.2330%
4	1711150	0.1458%
8	3409299	0.1742%

Test Phase: After the setup phase, the real test begins by starting the execution of the geth instances on the different machines. The clients emit transactions that cause simple transfers of values that do not require the execution of the EVM, with a predefined release time (`tx_interval`), while the miners start collecting transactions and creating blocks. The tests are stopped after a configurable time (`timeout`). For each configuration the tests are repeated a configurable number of times (`Runs`) because the probabilistic nature of the PoW algorithm and the interaction of many systems do not guarantee deterministic results.

The transaction release time is set to 50 ms in order to generate 20 transactions per second for each client and fill up the network processing capacity while avoiding flooding effects on low performance hardware. The test duration is set to 10 minutes thanks to the warm up already discussed in the setup phase. Since a single run would not be representative enough due to the non-deterministic components of such system, we deemed that 5 runs would be a good compromise between representativeness and resource availability.

We measured the throughput with different numbers of miners (1, 2, 4 and 8), fixing the number of clients at 16 with two different gas limits. The miner machine executes only one geth instance in miner mode, while the client machines execute two instances in client mode. For our tests we used the Scaleway platform[21].

We used 17 START1-S servers (1 master and 16 slaves), which are equipped with 2 X86 64 bit Cores and 2 GB of RAM, a 50 GB SSD and an internal bandwidth of 1 Gbit/s, so that the network would not be a bottleneck. Each server runs Ubuntu 16.04 LTS with geth version 1.8.11-stable. During the tests, the master acts also as bootstrap node, whose aim is to provide new nodes with contact information to other regular nodes that are already participating in the network. The two tests we conducted differ only in the gas limit in the genesis block. The reason for this is that we wanted to confirm that this value does indeed influence the number of processed transactions [9].

Results: We may now report a selection of interesting results obtained from the two tests that we described above.

[21]https://www.scaleway.com/

Table II
AVERAGE THROUGHPUT AND NUMBER OF MINED BLOCKS OVER 5 RUNS WITH LOW GAS LIMIT.

	1 miner	2 miners	4 miners	8 miners
Avg throughput	5.37	7.17	6.73	6.65
Avg blocks	24.00	37.80	30.02	30.00
Avg txs per block	133.77	114.69	133.61	133.15
Max throughput	7.18	7.76	8.17	8.67
Min throughput	3.68	6.83	5.91	4.91

Table III
AVERAGE THROUGHPUT AND NUMBER OF MINED BLOCKS OVER 5 RUNS WITH HIGH GAS LIMIT.

	1 miner	2 miners	4 miners	8 miners
Avg throughput	8.17	11.55	9.02	11.07
Avg blocks	33.60	27.20	37.20	29.80
Avg txs per block	145.25	262.98	157.15	226.32
Max throughput	12.55	17.50	17.77	12.525
Min throughput	6.82	6.83	6.83	7.14

Maximal throughput - Low Gas Limit: For this test we used a gas limit of $3,141,592$, which corresponds to approximately $150 \times 21,000$ gas units[22]. Table II reports the results we obtained for this experiment. We note that in this case we were limited to 10 transactions per second and 135 transactions per block. Indeed, the theoretical upper bound to the transactions per block is 150 (as noted earlier), but the intrinsic uncertainty of the system combined with the ability of the miners to perturb the gas limit can modify the number of transactions included in a block.

Maximal throughput - High gas limit: For this test we used a very large gas limit, set at $10,500,000$, which corresponds to $500 \times 21,000$ gas units. For reference, as of August 2018, the gas limit for the main Ethereum network is approximately 8 million, which corresponds roughly to $381 \times 21,000$ gas units. Table III summarizes the results obtained from this experiment. In this case, with two configurations, we could exceed 15 transactions per second.

VI. CONCLUSIONS

The Blockchain technology provides a way to assure a total order of transactions in a distributed system without relying on a trusted third party. This allows having an exact copy of the state in each node of the network, because each node starts from the same state and changes it according to the processed transactions, whose execution is deterministic. The main advantage of this technology with respect to traditional transition systems is its decentralized nature.

In Section III of this work, we have presented a decomposition of Ethereum's architecture in logical layers. The five-tiered architecture that we have discussed allows abstracting from the implementation details, and offers a conceptual organization that facilitates comparing different Blockchain proposals, while also enabling flexibility of interpretation, as typified by seeing the EVM as a vertical layer (cf. Fig. 1).

[22]A transaction with no EVM computation costs $21,000$ gas units.

The focus of Section IV and Section V centered on the analysis of the scalability of Ethereum. To this end, we empirically concluded that the current version of Ethereum reaches a maximum transaction throughput of approximately 15 tx/s, even if the transactions do not require computations. Interestingly, this value is shown to be *not* linear in the number of miners, but rather dependent on the block size and the block interval. We analyzed the reasons of this low transaction throughput with the aid of the Scale Cube [8]. In that analysis, we argued that the current version of Ethereum is not well versed in any direction of scaling. Thereafter, we categorized the scalability improvement proposals based on the axes they address.

This analysis showed that the most efforts of the community, with proposals such as Plasma and sharding, concentrate on the z-axis of the Scale Cube (cf. Section IV-E), thereby with a strategy that partitions along data or actions that are unique to individual request senders or receivers.

Sharding is a scaling strategy already seen in the database systems domain, that is, for systems that manage persistent data. Perhaps, this natural affinity between Ethereum and the databases may facilitate further improvements along the z-axis of the Scale Cube.

The result of our investigation maps directly to the scalability trilemma [13], which states that we cannot have at the same time scalability, decentralization and security. Assuming full security, we need to find a trade-off between scalability and decentralization. The current implementation of Ethereum provides security and decentralization and renounces scalability.

Proposals like the Proof-of-Stake [11] increase the transaction throughput. Yet, since the performance improvement does not depend on the number of nodes in the system, we cannot view the PoS as an improvement from the scalability standpoint. However, the PoS increments decentralization as it allows nodes with commodity hardware (as opposed to specialized one) to be validators.

Sharding and Plasma provide data partitioning, which allows parallel processing of transactions in different shards or child chains, thereby increasing the scalability of the system accordingly. However, the decentralization decreases in both proposals, since in Plasma no nodes have the blocks of the whole network (which causes some degree of mutual dependence), whereas in Sharding the same happens owing to the introduction of super-nodes. A technology like Blockchain cannot loosen up on security, which represents from the outset one of the central properties in this type of system, which adds to decentralization. The property of decentralization provides many advantages, but they are constrained by the costs to sustain it.

Although the hype around the Blockchain technology was – and continues to be – to be able to adapt and support any real-world application, the scalability of it was evident example of an open issue deferred to future consideration.

Now, the challenge is to increase the performance of the system overall without losing security, hence finding the best possible trade-off between decentralization and scalability, possibly creating different solutions for different applications.

The tests we presented in this work only concern transactions that do not involve smart contract execution, that is, that do not require the intervention of the EVM. The configuration setup discussed in Section V-A allows to reproduce as accurately as possible a real scenario of large scale distribution with a very limited amount of resources. Moreover, these parameters enable to decompose the actual problem isolating its single components, and, ultimately, understand how each piece of Ethereum affects its scalability.

One way to enhance our test framework is to allow the execution of smart contracts, possibly with different workloads, as suggested in [10]. The nature of the transaction might influence the transaction throughput of the system since they could involve computations, thus consuming more gas, and access part of the global state.

Apart from the maximum transaction throughput, we deem it interesting to measure other metrics of consequence to performance, for example, the latency, meant as the time interval between the transaction release time and its confirmation (that is, after a given number of blocks confirmed). Another promising future development for our test framework is to support heterogeneous clients. We noted the presence of various client implementations that can all coexist in the one and same system. Hence, the possibility to configure the network with different nodes, for both software and hardware, may increase the representativeness of the test conducted on our infrastructure.

References

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2018.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *Int'l Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[4] V. Buterin. (2018-04-12) A next-generation smart contract and decentralized application platform. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[5] X. Xu *et al.*, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE Int'l Conf. on Software Architecture (ICSA)*, April 2017, pp. 243–252.

[6] K. Croman *et al.*, "On scaling decentralized blockchains," in *Int'l Conf. on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[7] V. Buterin, "Ethereum 2.0 mauve paper," in *Ethereum Developer Conference*, vol. 2, 2016. [Online]. Available: https://cdn.hackaday.io/files/10879465447136/Mauve%20Paper%20Vitalik.pdf

[8] M. Abbott and M. Fisher, *The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise*. Pearson Education, 2009.

[9] A. Gervais *et al.*, "On the security and performance of proof of work blockchains," in *Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security*. ACM, 2016, pp. 3–16.

[10] T. T. A. Dinh *et al.*, "Blockbench: A framework for analyzing private blockchains," in *Proc. of the 2017 ACM Int'l Conf. on Management of Data*. ACM, 2017, pp. 1085–1100.

[11] Ethereum Foundation. (2018-08-09) CBC Casper FAQ. [Online]. Available: https://github.com/ethereum/cbc-casper/wiki/FAQ

[12] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, 2017.

[13] Ethereum Foundation. (2018-08-03) Sharding FAQs. [Online]. Available: https://github.com/ethereum/wiki/wiki/Sharding-FAQs

[14] G. Putnik *et al.*, "Scalability in manufacturing systems design and operation: State-of-the-art and future developments roadmap," *CIRP Annals*, vol. 62, no. 2, pp. 751–774, 2013.

[15] K. S. Ahluwalia, "Scalability design patterns," in *Proc. of the 14th Conf. on Pattern Languages of Programs*. ACM, 2007, p. 2.

[16] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, Sep. 2013, pp. 1–10.

[17] I. Eyal *et al.*, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[18] Y. Gilad *et al.*, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68.

[19] Ethereum Foundation. (2018-05-15) Node discovery protocol. [Online]. Available: https://github.com/ethereum/devp2p/blob/master/discv4.md

[20] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Int'l Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[21] Ethereum Foundation. (2018-05-25) JSON RPC API. [Online]. Available: https://github.com/ethereum/wiki/wiki/JSON-RPC#json-rpc-api

[22] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Int'l Conf. on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[23] MongoDB Inc. (2018-08-01) MongoDB Manual. [Online]. Available: https://docs.mongodb.com/manual/

[24] Ethereum Foundation. (2018-04-12) Ethereum Wire Protocol specification. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethereum-Wire-Protocol