SoK: Concurrency in Blockchain - A Systematic Literature Review and the Unveiling of a Misconception

Atefeh Zareh Chahoki[1], Maurice Herlihy [2], and Marco Roveri [1]

[1]University of Trento, Trento, Italy, `atefeh.zareh@unitn.it`, `marco.roveri@unitn.it`
[2]Brown University, Providence RI, USA, `mph@cs.brown.edu`

Abstract

Smart contracts, the cornerstone of blockchain technology, offer the prospect of secure and automated distributed execution. Given their involvement in managing vast volumes of transactions across various roles-including clients, miners, and validators- exploring concurrency is pivotal from multiple perspectives. This includes executing or validating transactions concurrently within each block, concurrent block processing across different shards, and concurrent competition among miners to select and persist transactions within blocks. Concurrency and parallelism represent a double-edged sword: Although increasing concurrency at any level benefits higher throughput, it also introduces potential risks such as race conditions, non-deterministic outcomes, and vulnerabilities such as deadlock and livelock.

This paper presents the first survey of concurrency in smart contracts, offering a comprehensive systematic review of the literature, and organizing it into distinct dimensions. Firstly, it establishes a comprehensive taxonomy of existing levels of concurrency within blockchain systems and discusses currently proposed solutions for incorporation into future blockchain iterations. Secondly, given the imperative for correctness and security in massively distributed processing infrastructures, this paper examines concurrent operations' vulnerabilities, potential attacks, and countermeasures. Critically, we analyze the assumptions used by each category and highlight a flaw in the concurrency assumption in a major category. This misconception has led to misinterpretations in important research bodies. Thus, this work aims to correct long-standing misconceptions within the field and steer future research toward more genuine assumptions. Finally, we clarify the directions for future research by identifying gaps within each category, thus contributing to the advancement of the blockchain community.

1 Introduction

Smart contracts are programs that can be stored and executed on a blockchain and that automate the execution of agreements according to predefined logic between untrusted parties [17]. Due to their tamper-proof nature and immutability of state transitions, they are ideal for enabling and facilitating trustless transactions in various domains. Deterministic execution is essential for smart contracts to ensure the predictable outcome of a transaction based solely on its inputs and the current state of the blockchain. This predictability is crucial for reliable decision-making and for avoiding unintended consequences. The concept known as the "Blockchain impossibility triangle" [18] or "Blockchain Trilemma" [19] suggests that a blockchain system cannot simultaneously achieve optimal scalability, decentralization, and security. Although this trilemma is not a formal proof within blockchain architecture, it effectively encapsulates the primary challenges blockchain systems face as observed in current implementations and industry practices. No blockchain system excels in all three areas. Traditionally, most blockchain systems have prioritized decentralization and security over scalability [73], which fundamentally limits their performance. Similarly to how different distributed systems must balance consistency and availability due to the CAP theorem, blockchain systems must also navigate trade-offs between scalability, decentralization, and security depending on the scenario. Scalability should not always be treated as the least important factor and sacrificed as a result.

One approach to addressing the scalability challenge is concurrency. This concept can be applied at various levels of the blockchain infrastructure, such as sharding, which enables transactions to be processed concurrently across different shards, or executing smart contracts within each shard and for each block by validators concurrently in a multi-core setting. Other strategies explored in this survey also leverage concurrency to enhance scalability.

However, concurrency introduces challenges related to synchronization, consistency, race conditions, deadlocks, and livelocks. These challenges can result in security vulnerabilities and unexpected behavior in smart contracts. Therefore, ensuring that these issues are addressed is critical for any concurrency solution, as discussed in this survey.

This survey aims to provide a comprehensive and systematic review of all types of concurrency issues in smart contracts. It assesses the validity of the assumptions and breaks them down into a taxonomy and a more detailed organization. In each area, the future open questions are elaborated on and presented. The study also aims to examine weaknesses, attacks, and countermeasures in the field, critically analyze assumptions, and correct long-held misconceptions to align future research with real assumptions.

This paper presents several key contributions: (1) To the best of the authors' knowledge, it provides the first comprehensive survey of concurrency in smart contracts, offering a systematic review of a wide range of current literature to serve as an organized and insightful resource for researchers; (2) a detailed taxonomy is introduced, classifying various concurrency aspects into categories and organizing them based on their approaches and goals, whether implemented on existing blockchains or proposed for future developments; (3) open questions and emerging trends in all defined categories are highlighted, offering valuable guidance for future research; and (4) the flawed as-

sumption of concurrent execution of transactions in Category Six of the presented taxonomy is critically examined and discussed, supported by sufficient references. All materials and code required to conduct this survey are publicly available on GitHub [1].

The remainder of this paper is organized as follows: Section 3 provides the main background concepts used throughout the paper. The methodology of the systematic review is presented in Section 4. Section 5 organizes the concurrency perspectives into three categories. In Section 6 a critical analysis reveals a misunderstanding in the third category.

2 Problem Statement And Motivation

Existing blockchain systems require that every node in the network maintain a complete replication of the transaction history and ledger states to ensure the total order of transactions, execution integrity, and data provenance. These data structures grow significantly over time, becoming prohibitively large. For instance, as of July 2024, the Bitcoin ledger is approximately 550GB [2], while the Ethereum ledger exceeds 18.5TB [23]. To address this, most blockchain nodes maintain a smaller and compact index known as *validation states*, which are sufficient to validate transactions. However, even these validation states can be substantial in size, such as Ethereum'cas validation states, which are around 1,120GB[23]. Additionally, blockchain nodes must locally replay all transactions based on these replicated states, incurring significant storage and computation costs that limit system scalability. This cumbersome stateful data also undermines system security and robustness by centralizing the network, as fewer nodes can handle such large data volumes.

To mitigate these issues, one approach is sharding, which partitions the blockchain into multiple parallel chains, each managed by a subset of nodes. This reduces storage and computation duplications among different shards. However, sharding only alleviates the problem by a constant factor, as nodes within each shard still duplicate storage and computation. Moreover, sharding introduces new challenges, such as cross-shard transaction processing and vulnerability to attacks by slowly adaptive Byzantine adversaries. Shardening is studied in Section 5.2.

Another approach is the use of light nodes that store only block headers rather than full states. Although light nodes can follow consensus protocols, they cannot independently verify transaction validity, failing to address centralization due to state maintenance burdens.

The stateless blockchain concept has recently emerged as an off-chain scaling approach. This method moves ledger states and transaction executions off-chain to a subset of nodes, thereby reducing on-chain load. However, existing stateless blockchain systems are primarily designed for cryptocurrencies and face limitations when applied to general-purpose use cases involving smart contracts. Developing a general-purpose stateless blockchain that supports smart contracts presents several challenges. Transactions involving smart contracts can contain arbitrary logic, requiring novel proof techniques to ensure the integrity of off-chain executions. Smart contract transactions

---

[1] https://github.com/Atefeh-Zareh/concurrency-in-smart-contracts

also introduce read-and-write sets of varying sizes, necessitating additional design for on-chain commitment updates. Furthermore, existing methods for cryptocurrency exchange offer limited support for parallel transaction executions. To enhance system throughput, new transaction processing methods are needed to validate and commit concurrent transactions from an asynchronous network, despite the stateless design. This category of off-chain solutions is studied in Section 5.6.

3 Background

This section briefly reviews the concepts related to concurrency and blockchain used in this paper.

3.1 Concurrency concepts

Concurrency refers to the ability of a computer program to execute multiple instructions or processes simultaneously. In distributed systems, concurrency can introduce challenges related to:

- **Consistency**: Maintaining data integrity across different replicas or copies in a distributed system.

- **Synchronization**: Ensuring that multiple processes access shared resources in a coordinated manner to avoid data inconsistencies.

- **Race Conditions**: Scenarios where the outcome of a program depends on the unpredictable timing of concurrent processes.

- **Nondeterminism**: Parallel systems commonly demonstrate this trait due to competition for communication resources. A system shows nondeterminism when two identical copies of it can act differently even with the same inputs upon which wins the race that results in an untestable trait of these systems. [54]

- **Deadlock**: is one of the most common issues in a parallel system, when no component can progress due to mutual waiting for communication, a concurrent system is deadlocked. A classic example is the 'five dining philosophers', where each philosopher requires both neighboring forks to eat, leading to a deadlock if all philosophers simultaneously pick up their left-hand fork and they deadlock and starve to death. One of the main reasons for deadlock is competition to acquire the resources [54].

- **Livelock**: The cause of livelock, is called *divergence* which means a program performs infinite unbroken sequences of internal actions and never interacts with its environment, such as infinite loops in programs. Although operationally and theoretically deadlock and livelock are different, they appear similar from the users' perspective because no response is received. However, livelock is worse since users may observe internal activities and mistakenly expect an eventual output [54].

4

3.2 Smart contracts

The *blockchain* is a foundational design pattern to facilitate pure peer-to-peer distributed computation. Initially implemented by Bitcoin in 2009 as a cryptocurrency [48], it has since evolved into a robust infrastructure capable of executing a wide range of generalized functionalities beyond cryptocurrency. Ethereum [16], in particular, pioneered this expansion in 2013 by introducing the *Ethereum Virtual Machine (EVM)*, a Turing-complete state machine that handles both deployment and execution of codified arbitrary business logic scripts named *smart contracts*. All blockchain networks maintain securely shared data named *ledger* through a *consensus protocol* facilitated by volunteer operators named *miners* and *validators*. The transactions announced by users are initially stored in the *mempool* (short for "memory pool"), a temporary storage area where pending transactions wait to be included in the next block. Miners select transactions from the mempool that offer higher fees first, as this maximizes their rewards and persists them into the next block if the transaction is valid. This validity check includes the correctness of transaction format, having a sufficient gas fee, nonce verification, signature verification, balance check, double-spending prevention, compliance with consensus rules, and regarding smart contracts ensuring that the transaction can execute without error. Then miners extend the blockchain by one block in each *block interval*, which varies across different networks such as 10 minutes in Bitcoin or 12 seconds in Ethereum. Smart contracts introduce the concept of *gas*, irrespective of the consensus protocol they utilize. Gas represents the computational effort and cost required to execute operations within smart contracts to compensate for the resources used to execute smart contracts. It prevents infinite loops, restricts resource consumption, and maintains the network's efficiency and security. Smart contracts are written in various programming languages tailored to specific blockchain platforms. Solidity is the predominant language for Ethereum; Vyper aims to provide simplicity and security; Michelson is used on Tezos; and Move was developed for Diem (formerly known as Libra).

3.3 Proof of Work (PoW) & Bitcoin

Proof of Work (PoW) is a consensus protocol that is mainly introduced in Bitcoin. Participants, known as *miners*, gather newly broadcasted and not yet persisted transactions and add them in a *block* data structure if they are *valid*, then compete to solve a cryptographic puzzle, named *mining*, to add the block to the ledger. One part of the block is its *previous block hash*, which makes this data structure *tamper-proof*. The first miner to solve the puzzle earns the right to add the next block of transactions to the blockchain and win rewards. Rewards are in the form of the newly *minted* cryptocurrencies in that block and the *fee* and gas of the included transactions as the incentive. PoW is the original consensus mechanism used by Bitcoin [48] and several other cryptocurrencies, but because it is energy-intensive, it is being replaced by others such as Ethereum.

3.4 Proof of Stake (PoS) & Ethereum Mechanism

This section provides an overview of Proof of Stake (PoS), one of the most widely adopted consensus algorithms in blockchain infrastructures, including Ethereum. This section also briefly explores Ethereum's execution mechanism, highlighting its key components and processes.

In PoS, a node is selected as the proposer using methods like computational contests, predetermined sequences, or random algorithms, as seen in Ethereum. The proposer compiles a block by selecting a set of transactions and shares it with other nodes, known as attestors. Each attestor independently validates the transactions in the block, rejecting it if any discrepancies or invalid transactions are found.

Ethereum transitioned to PoS with its Ethereum 2.0 upgrade in 2022, aiming to enhance security, reduce energy consumption, and support new scaling solutions compared to the prior Proof of Work (PoW) architecture. In PoS, participants, referred to as *validators* or *nodes*, must deposit 32 ETH into a deposit contract, thereby *staking* it as collateral [1]. Validators are responsible for ensuring the validity of new blocks and occasionally for proposing and propagating them. Misbehavior, such as proposing multiple conflicting blocks or sending invalid attestations, results in penalties, including partial or total loss of staked ETH. To join as a validator, a user must pass through an activation queue, which limits the rate of new validators joining the network. This mechanism ensures system stability, efficient consensus, and protection against resource strain and Sybil attacks, where a single entity creates multiple identities to gain disproportionate influence over the network.

In Ethereum's PoS system, block timing follows a fixed schedule rather than being influenced by mining difficulty as in PoW. Time is divided into *slots* of 12 seconds and *epochs* comprising 32 slots, corresponding to 6.4 minutes. During each slot, a validator is pseudo-randomly selected using RANDAO to propose a new block and broadcast it to the network. Simultaneously, a randomly chosen committee of validators evaluates the proposed block's validity. These committees distribute the workload, ensuring that all active validators participate within each epoch without needing to act in every slot [1].

3.4.1 Transaction Process

The process of executing a transaction in Ethereum PoS involves several steps. A user creates and signs a transaction with their private key via a wallet or library, essentially making a request to a node through the `Ethereum JSON-RPC API`. The transaction is submitted to the *execution client* part of that node, which verifies its validity, ensuring that the sender has sufficient ETH and has signed the transaction correctly. Valid transactions are added to the *local mempool* (a list of pending transactions) and broadcast over the execution layer gossip network. Advanced users may bypass public broadcasting and forward their transactions to specialized block builders, such as Flashbots Auction, to maximize profits through *Maximal Extractable Value (MEV)* [1].

The selected validator for the current slot is responsible for updating the global state. Nodes run three key software components: the *execution client*, the *consensus client*, and the *validator client*. The execution client collects transactions from the local

mempool, executes them locally to generate state changes, and compiles them into an *execution payload*. This payload is passed to the consensus client, which incorporates it into a *beacon block*. The beacon block also contains metadata such as rewards, penalties, slashings, and attestations, enabling consensus on the blockchain's state and head of the chain [1].

Other nodes receive the new beacon block via the consensus layer gossip network and pass it to their execution clients for local validation of the proposed state changes. The validator client attests to the block's validity and confirms that it builds upon the chain with the greatest attestation weight, as determined by the fork choice rules. Once validated, the block is added to the local database of each node.

3.4.2 Transaction Finality

A transaction achieves *finality* when it is part of a block that cannot be reverted without burning a significant amount of staked ETH. Finality is managed through *checkpoint* blocks, which occur at the start of each epoch. Validators vote on checkpoint pairs, and if a pair receives votes from at least two-thirds of the total staked ETH, the target checkpoint becomes *justified*, and the source checkpoint is upgraded to *finalized*. Reverting a finalized block would require an attacker to control and sacrifice at least one-third of the total ETH staked across all participants in the network. To prevent an attacker from stalling finality, Ethereum employs an *inactivity leak*, which penalizes validators voting against the majority when finality fails for more than four epochs. This ensures the majority regains a two-thirds stake to finalize the chain.

3.4.3 Ethereum's Layered Architecture

Ethereum's architecture is organized into several layers, each responsible for specific functions that contribute to the overall operation and scalability of the network. The **execution layer** handles transaction processing, smart contract execution, and state changes. This layer is responsible for bundling transactions into *execution blocks* and updating the global state. Above the execution layer is the **consensus layer**, also known as the *Beacon Chain*, which is responsible for maintaining network consensus, coordinating validators, and ensuring security. The *beacon block* is the key data structure in this layer, as it organizes validator attestations and records the consensus state of the network. Ethereum also integrates a **data availability layer**, which ensures the reliable retrieval of off-chain data, particularly for layer 2 solutions. Lastly, the network utilizes **layer 2 scaling solutions**, such as **Optimistic Rollups** and **zkRollups**, which offload transaction processing to achieve greater throughput while maintaining security through periodic anchors to the mainnet. This modular, layered architecture enhances Ethereum's scalability and security while enabling the development of decentralized applications and solutions.

3.4.4 Ethereum Networks and Their Purposes

Ethereum operates across several networks, each tailored for specific purposes within its ecosystem. The **mainnet** is the primary public network where real transactions,

decentralized applications (dApps), and smart contracts are executed, using ETH with actual monetary value. For development and testing, Ethereum offers **testnets** such as **Goerli** and **Sepolia**, which simulate mainnet conditions without risking real assets, with Goerli widely adopted for advanced testing and Sepolia preferred for lightweight development. **Private networks** are custom deployments used by organizations or developers for controlled experimentation and application testing. To address scalability, Ethereum integrates **layer 2 networks** like **Optimism** and **Arbitrum** (Optimistic Rollups) and **zkSync** and **StarkNet** (zkRollups), which reduce transaction costs and increase throughput by offloading computation from the mainnet while retaining security guarantees. Additionally, emerging **data availability solutions** like **EigenLayer** ensure that off-chain data for rollups can be reliably retrieved, further enhancing Ethereum's scalability and modularity.

3.4.5 Terminology and Roles in Ethereum PoS

In Ethereum's PoS architecture, the terms *validator* and *node* are used interchangeably. Each node performs multiple roles: proposing blocks when pseudo-randomly selected for a slot, attesting to other validators' blocks within an epoch as part of a committee, and validating transactions in each slot. These roles are not distinct among nodes but represent different responsibilities within each node.

However, various terminologies in the literature might appear to suggest different roles. For instance, some references distinguish *miners* and *validators*, where miners propose new blocks and validators verify them [22]. Others, such as [68], use terms like *proposer* and *attestor* within a proposal-attestation paradigm. In this context, proposers select transactions and broadcast blocks, while attestors validate the blocks. In Ethereum, these distinctions are purely functional, as all nodes perform these tasks at different times.

In this survey, we adhere to the Ethereum convention, referring to participants as *validators*. For consistency and clarity, references to other terminology will include appropriate explanations, ensuring that readers are not misled by differing conventions.

3.5 Hyperledger Fabric

3.5.1 Nodes in Hyperledger Fabric

In Hyperledger Fabric, the nodes serve as communication entities within the blockchain network. Unlike many blockchains where nodes operate in a peer-to-peer manner, nodes in Hyperledger Fabric have distinct roles. There are three types of nodes:

- **Client**: The Client is operated by the end user and is responsible for submitting transaction proposals to the Endorser Peer and broadcasting the proposals and responses to the Orderer.

- **Peer**: Peers are primarily tasked with executing chaincode to read from and write to the ledger. All Peers act as committing peers (Committers), maintaining the ledger's state. Additionally, peers can serve as endorsing peers (endorsers) when

an application makes a transaction endorsement request. This endorsement role is dynamic; otherwise, Peers function as regular committing peers.

- **Orderer**: The Orderer nodes collectively form the ordering service. In Hyperledger Fabric, which is a distributed system, each node maintains a copy of the ledger. The ordering service ensures the consistency of the ledger by ordering transactions into blocks.

3.6 Other Consensus Protocols

There are two categories of consensus protocols: Proof-based and Leader-based [73].

Byzantine fault tolerance (BFT) is a critical property in distributed system design. It refers to the system's ability to maintain its correct operation even when some components fail or behave maliciously. In blockchain networks, Consensus protocols are mechanisms used by distributed systems to achieve agreement on a single data value among multiple nodes or participants, even in the presence of faulty or malicious actors. This property is essential to maintain the integrity and security of the distributed ledger. Various consensus algorithms have been developed to achieve Byzantine fault tolerance in blockchain systems, each with its own trade-offs in terms of efficiency, scalability, and security. The following reviews some of the main types of consensus protocols, excluding PoW and PoS, which are discussed later [37]:

- **Delegated Proof of Stake (DPoS):** DPoS is a variation of PoS where token holders vote for a select few delegates who are responsible for validating transactions and adding them to the blockchain. EOS and TRON are examples of blockchain platforms that use DPoS.

- **Proof of Authority (PoA):** In PoA, consensus is achieved by a set of approved validators, typically known entities or organizations. Validators are responsible for creating new blocks and maintaining the blockchain. PoA is used in networks where a high degree of trust among participants is assumed, such as private or consortium blockchains.

- **Practical Byzantine Fault Tolerance (PBFT):** PBFT is a consensus algorithm designed to work in asynchronous networks with a certain number of faulty or malicious nodes. It ensures that all honest nodes reach a consensus on the order of transactions. PBFT is used in permissioned blockchains and distributed systems like Hyperledger Fabric.

- **Raft:** Raft is a consensus algorithm designed for fault-tolerant distributed systems. It elects a leader node responsible for managing the replication of the log among other nodes. Raft is often used in distributed databases and systems where simplicity and ease of understanding are prioritized.

- **Directed Acyclic Graph (DAG):** DAG-based consensus protocols, such as Tangle (used by IOTA) and Hashgraph, do not rely on blocks or miners. Instead, transactions are directly linked to each other in a graph-like structure, and consensus is achieved through a voting process based on transaction approval.

9

- **Proof of Vote (PoV):** Proof of Vote (POV) [38] represents a consensus algorithm designed specifically for consortium blockchains where PoW falls short. In this model, consensus is orchestrated by distributed nodes overseen by consortium partners, reaching decentralized decisions through voting-based arbitration. The primary concept involves assigning distinct security identities to network participants, enabling block submission and verification via voting among these entities, eliminating the need for third-party intermediaries or reliance on unregulated public awareness. In contrast to the fully decentralized PoW consensus, POV offers controllable security, convergence, and reliability, achieves transaction finality with a single block confirmation, and ensures swift transaction verification with minimal delay.

3.7 Performance in smart contracts

Table 1 shows different cryptocurrencies in the first column, the transaction speed as TPS in the second column, which stands for "transactions per second" and the last column shows the "average transaction confirmation time" of these cryptocurrencies [30]. The transaction verification process for cryptocurrencies is very slow and does not match the performance of traditional payment systems such as VISA, which handles an average of 10,547 TPS and can peak at 56,000 TPS and Paypal with a TPS of 1,700. TPS for Bitcoin is almost 7 and other TPSs for cryptocurrencies are listed in the second column on this paper. The calculation of the TPS for the cryptocurrencies in the last year is available in this paper git path.

Numerous approaches have been proposed to increase transaction speed and scalability by enhancing concurrency efficiency in blockchain technology. In Section 5, we will review these approaches, focusing on how concurrency is utilized at different levels of the blockchain to achieve improved performance.

Table 1: transaction speed of different cryptocurrencies

Cryptocurrency	TPS	Time
Bitcoin	$3-7$	60 min
Ethereum	$15-25$	6 min
Ripple	1500	4 sec
Bitcoin Cash	61	60 min
Stellar	1000	$2-5$ sec
Litecoin	56	30 min
Monero	4	30 min
IOTA	1500	2 min
Dash	48	$2-10$ min

4 Methodology

This section outlines the questions to be addressed, along with the source dataset and the query used to extract the relevant literature for this systematic review. Subsequently, it presents the exclusion criteria.

4.1 Research questions

This study aimed to address several crucial research inquiries:

(RQ1): What are the distinct concurrency aspects associated with smart contracts and their blockchain infrastructure?
- This question focuses on gaining insights into all types of concurrency aspects within the blockchain ecosystem and organizing them into categories.

(RQ2): What are the existing security threats and countermeasures available in the domain of blockchain concurrency, and how do they address the identified concerns?
- This investigation aims to identify the techniques currently utilized to address concurrency challenges in smart contracts in a detailed representation. It categorizes the literature based on whether the aspect already exists in the blockchain infrastructures or is proposed for further iterations. In addition, it examines whether the study's focus is on introducing vulnerabilities, attacks, or security solutions, and analyzes the approaches taken to address these issues.

(RQ3): What are the potential research gaps of current solutions that require attention?
- The purpose of this inquiry is to organize the open questions and trends in all these categories. These identified threats can serve as guidelines for future research in this field, directing attention to areas that require further investigation due to the absence of proposed solutions or the limitations of existing approaches.

4.2 Source databases and search query

This study used Scopus[2] as the primary source to identify relevant publications in this field. Scopus provides a vast repository of scientific literature with over 27,950 active titles, 90.6+ million records, and from more than 7,000 publishers. Scopus is continuously expanding its coverage of conference material primarily for the subject domains of publishers and societies of engineering and computer sciences [59]. In addition, Scopus offers a powerful search engine specifically designed for academic exploration. Although other databases such as Google Scholar, Web of Science, and IEEE Xplore exist, the extensive coverage and robust search capabilities of Scopus make it a sufficient choice for this particular research.

The search query utilized for all repositories can be found in Listing 1:

In line 1, TITLE-ABS-KEY indicates that the search query is applied to all the title, abstract, and keyword sections of the repositories. The primary objective of the search is to identify research works that specifically address concurrency solutions 4) in blockchain infrastructures (line 2). The query utilized the wildcard * in 'concurren*' and 'synchroni*' to encompass all variations within the word families of concurrency and synchronization. The query targeted all papers in the English language (line 5) and excluded document types of conference review (cr), abstract report (ar), book (bk),

---

```
1  TITLE-ABS-KEY (
2    ( "smart contract" OR "Ethereum" OR "solidity" OR "Hyperledger" )
3    AND ( "concurren*" OR "race" OR "parallel" OR
4         "multithreaded" OR "synchroni*" OR "transaction ordering
               dependancy" OR "front running attack" OR shardening)
5    AND ( LANGUAGE( "English" ) )
6    AND NOT( DOCTYPE ( "cr" ) OR  DOCTYPE ( "ab" ) OR DOCTYPE ( "bk" ) OR
         DOCTYPE ( "ch" ) OR DOCTYPE ( "cr" ) OR DOCTYPE ( "re" ) OR DOCTYPE
         ( "sh" ) )
```

Listing 1: Search query.

book chapter (ch), conference review (cr), review (re) and short survey (sh), respectively, encoded on line 6. The described query execution in May 2024 returned 237 unique papers before the pruning process.

4.3 Inclusion and exclusion criteria

Table 2 lists the inclusion and exclusion criteria that are applied in this current study.

Table 2: Inclusion and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
• Research studying any concurrency aspects in smart contracts. • Studies that address at least one of the designated research inquiries. • Relevant studies identified through snowballing techniques from previously chosen literature.	• Theses, books, survey-based studies, or patents. • Non-peer-reviewed research. • Duplicate studies. • Studies lacking relevance to the designated research questions.

4.4 Screening Results

Figure 1 illustrates the percentage of relevant articles discovered during the screening process. Additionally, a sub-pie chart categorizes the non-relevant papers into different domains, highlighting the areas that were identified and filtered out from the main set of papers under consideration.

Figure 2 depicts the total number of citations for the relevant articles, categorized by their year of publication.

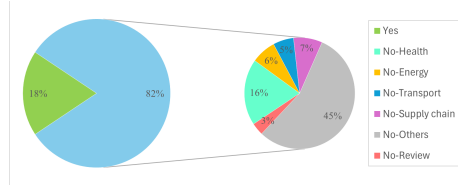Figure 3 shows the number of relevant articles published each year.
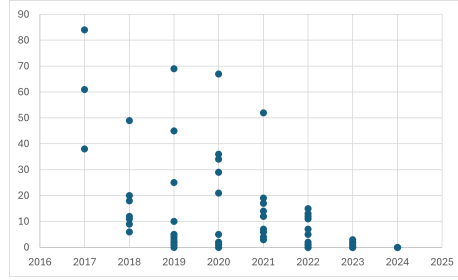
Figure 1: Screening results

Figure 2: Paper citations per year

5 Concurrency in smart contracts: a taxonomy

Three main categories of concurrency approaches for smart contracts have emerged that are represented in this section.

5.1 Category 1: Concurrent Consensus

This category of solutions focuses on consensus protocols and believes that to scale blockchain systems, the fundamental structure of consensus needs modification. They propose novel consensus protocols to enhance concurrency in blockchain infrastructure aiming to improve performance.

One notable research in this category is done by Hazari and Mahmoud [29], who introduced *parallel PoW* based on parallel mining rather than solo mining. This approach eliminates simultaneous efforts of miners to solve a specific block by introducing a new role as a *manager*, along with processes for manager selection, work distribution, and a reward system. The introduction of a manager role does not compromise the decentralized nature of blockchain. This solution was implemented in a test environment mimicking Bitcoin's PoW features and tested across various scenarios with different difficulty levels and numbers of validators. The results indicate a 34% improvement in PoW scalability. This protocol aims to increase transaction processing speed, potentially reducing energy consumption. This research is extended by these authors in [30] which evaluated a cloud-based environment. The future plans for this research include evaluation against the 51% attack, evaluation on the Bitcoin testnet, refinement of the reward system, and evaluation of real-time network energy consumption, although it is expected to end reduced energy consumption due to speeding up transaction process-
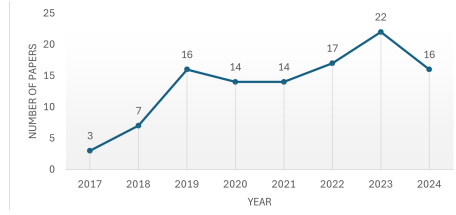
Figure 3: Paper number per year

ing.

[39] introduces an approach to execute smart contracts in parallel and asynchronously. Unlike traditional methods, this approach minimizes coordination efforts and eliminates risks such as livelocks, even with small groups. This paradigm is implemented in two ways: first, enhancing Ethereum to support parallel and asynchronous execution seamlessly, without the need for hardforks. Second, introducing SaberLedger, a new public and permissionless blockchain, demonstrating its capabilities through a prototype implementation.

In [35], the authors introduce BDLedger (Big Data Ledger), a blockchain-like distributed ledger designed to achieve a nearly linear throughput scaling. BDLedger features a unique consensus mechanism called *Random Witness Consensus* and uses a DAG-based block structure. To support this scaling, the system makes two key trade-offs: 1) it does not establish consensus on transaction order but focuses only on transaction content, which ensures that transactions do not conflict and can be processed in parallel; 2) it limits the nodes that store block content to a fixed number of random nodes, which helps control storage and bandwidth costs. The study demonstrates that BDLedger can scale throughput from approximately 20,000 TPS with 10 nodes to about 160,000 TPS with 100 nodes.

[11] focuses on improving PBFT, a consensus-based algorithm used in the Tendermint blockchain [36]. The paper introduces a lock-free algorithm for the blockchain, enhancing concurrency by allowing the proposal and voting phases to occur concurrently while keeping the commit phase sequential. This modification aims to mitigate the negative impacts of locks, thereby enabling greater parallelism within the blockchain system.

[13] focuses on concurrency in the consensus algorithm level to avoid a throughput decrease by node numbers. They have proposed a more sufficient consensus algorithm based on PoV (read more in 3.6) named *Parallel Proof of Vote (PPoV)*. In this protocol, concurrency is integrated into the consensus cycle, enabling numerous nodes to generate blocks simultaneously. Its research findings and experimental data indicate that PPoV outperforms traditional BFT consensus by a factor of 2-5, particularly when the number of nodes ranges between 4 and 100.

5.2 Category 2: Sharding

Sharding, originally a database technique for splitting large datasets into smaller parts across multiple storage systems, improves efficiency by reducing server load. In blockchain,

14

sharding was first introduced in 2016 [43] as a method to enhance scalability and transaction throughput. The protocol *ELASTICO*, proposed for permissionless blockchains, scales transaction rates almost linearly with available computational power. It efficiently uses network messaging and tolerates Byzantine adversaries up to one-fourth of the computational power. ELASTICO achieves secure partitioning of the mining network into smaller committees, each handling a distinct set of transactions or *shards*. It was the first secure sharding protocol designed to operate in the presence of Byzantine adversaries.

Since then, sharding in blockchain has become one of the most advanced and specialized areas in the context of concurrent executions on blockchain networks. Sharding, in particular, has been extensively examined in several key surveys, and readers are encouraged to consult these sources for in-depth analysis. This survey aims to address critical gaps in the concurrency domain, which remains underexplored. For a more detailed overview of recent developments in the specific area of sharding, see [70], [40], [28], [4], [14], [62], [52].

5.3 Category 3: Intra-block Concurrent Processing

This approach acknowledges the sequential execution of transactions within a block by miners or validators in modern cryptocurrency systems like Ethereum and aims to address the limitations of serial execution of both inter and intra-Smart Contracts, which result in inefficiencies in system throughput and resource utilization.

5.3.1 Speculative

To overcome this challenge, [22] presented a novel approach to enable the parallel execution of smart contracts by miners and validators. This approach utilizes established concepts from Herlihy et al.'s distributed computing approaches *software transactional memory (STM)* [33] and *transactional boosting* [32]. The speculative approach has two strategies. In the first strategy, "execute then order," the leader speculatively executes smart contracts (SCs) in any arbitrary order, generates a corresponding schedule, and then sends this schedule to the validators, who replay the transactions and vote on the execution outcome. In the second strategy, "order then execute," the leader predetermines the execution order of the SCs. Validators then speculatively execute the SCs in parallel, but they ensure that the final outcome reflects the predetermined order.

The first one is "order then execute," in which the leader chooses an order in which the SCs should appear to execute. The validators speculatively execute the SCs but ensure that the result seems to be executed in that order. The second strategy is "execute then order," in which the leader executes the SCs speculatively in an arbitrary order, generates a schedule, and sends that schedule to the validators who replay and vote on that execution. These strategies are discussed in more detail in the following.

Order then execute. Miners execute smart contracts *speculatively* in parallel, allowing non-conflicting contracts to proceed concurrently and generating a serializable concurrent *schedule* for a block's transactions. Miners encode this schedule as a deterministic *fork-join program*, facilitating parallel re-execution by validators. Despite its efficacy in increasing transaction throughput, this approach is based on the sequential

nature of transaction execution within a block and does not introduce any uncertainty in transaction executions. Experimental results demonstrate significant speedups for both miners and validators, with a 1.33x speedup for miners and a 1.69x speedup for validators achieved with just three concurrent threads. Additionally, the paper presents a prototype implementation and discusses the potential for future advancements in multi-threading support and programming language design to further improve throughput and concurrency control in smart contract execution. Furthermore, the paper highlights the applicability of the proposed mechanisms in permissioned blockchain systems and outlines future research directions, including support for multithreading in the Ethereum virtual machine and advancements in programming language support for smart contracts to maximize throughput while avoiding concurrency pitfalls [22].

This approach has been experimentally evaluated in [55] to represent the potential benefits of speculative techniques for the execution of Ethereum smart contracts in parallel, using historical data to estimate their effectiveness. Transaction traces from sampled blocks on the Ethereum blockchain are replayed over time using a simple speculative execution engine, with miners attempting to execute all transactions in parallel and rolling back those causing data conflicts. Validators follow the same schedule as miners. The findings reveal that even simple speculative strategies yield notable speed-ups, with estimated speed-ups starting at approximately 8-fold in 2016 and declining to about 2-fold by the end of 2017, as transaction traffic increased. Moreover, the study identifies that a small set of contracts is responsible for a significant portion of data conflicts resulting from speculative concurrent execution, named *storage hot-spots*. Several observations emerge from the study, including the importance of distinguishing between reads and writes to reduce conflict rates, limited additional benefits from more aggressive speculative strategies, and the potential for accurate static conflict analysis to yield modest benefits. Furthermore, increasing the number of cores in the simulated virtual machine improved speed-ups, but with little improvement beyond 64 cores. The study suggests directions for future research, including incentivizing contracts that produce fewer data conflicts and extending the virtual machine to support common commutative operations. In conclusion, while simple speculative strategies can produce significant speed-ups, these diminish as transaction rates and conflict rates increase. The study underscores the need to focus on reducing conflict rates to further increase parallelism in Ethereum-style smart contract execution, potentially through improved recognition of commuting operations at the semantic level and investigating the effects of intrinsic data types on contention.

Execute then order. [21] addresses the challenge of integrating black-box concurrent data structures into existing STM frameworks, focusing on optimistic transactional usage. It introduces conflict abstractions to define data-structure conflicts in terms of commutativity separately from object implementation, enabling efficient cooperation with generic STM runtimes. Additionally, shadow speculation allows individual transactions to make private speculative updates to highly concurrent black-box objects, facilitating opaque speculative updates that can be later dropped or atomically applied. These concepts are realized in a new transactional system called ScalaProust, built on ScalaSTM, supporting objects of arbitrary abstract type and integrating with underlying STM conflict-detection mechanisms. The paper's contributions include conflict abstractions, shadow speculation, ScalaProust system implementation, and ex-

perimental validation demonstrating competitive scalability with existing specialized approaches. However, the described mechanisms currently do not support mixtures between transactional objects and STM-managed read/write operations. Future directions include integrating pessimistic and optimistic treatment of black-box ADTs with standard STM memory operations, improving performance, and utilizing conflict abstractions for STM support of "pure writes" and automatic verification techniques for synthesis.

5.3.2 Deterministic

[68] presents a two-stage approach for efficiently executing Blockchain transactions in parallel by adopting deterministic concurrency control. The paper addresses the inefficiencies of generating dependency graphs, which are often time-consuming and require re-executing transactions multiple times. Instead of building a large dependency graph, the authors partition transactions into *batches* to avoid inter-batch conflicts, thus reducing the complexity of computing partial orders. The proposed *deterministic concurrency control (DVC)* method enables high parallelism by solving an equivalent Min-VWC problem (minimum vertex weighted coloring) [64], which helps in finding an approximate optimal partial order without the need for re-execution of transactions. The authors demonstrate the effectiveness of their approach through experimental results, which show a significant reduction in the costs of computing the partial order and achieving a high degree of parallelism compared to existing solutions. This work offers a promising solution to improve the efficiency of concurrent Blockchain transaction execution, especially in the context of modern hardware utilization.

5.3.3 Partitioning

This approach proposes a scalable smart contract execution scheme for blockchain-based systems, aiming to overcome the limitations of serial processing and improve system throughput. Using the decentralized nature of blockchain technology, the scheme enables the parallel execution of multiple smart contracts, thus enhancing the system's capability to handle a large volume of transactions. The study carried out in [25] approach employs a divide-and-conquer strategy, using a fair contract partition algorithm based on *integer linear programming (ILP)* to partition smart contracts into subsets. Each subset is then randomly assigned to a group of users, named sub-committee. Participants within each sub-committee exclusively handle a portion of smart contracts, enabling the simultaneous execution of various smart contracts. The contributions of this work include the development of a scalable execution scheme, theoretical analyses demonstrating its correctness and security, and simulations showcasing its practicality. The study evaluates the efficiency of the proposed scheme using data from existing smart contract systems, demonstrating its scalability and improved efficiency compared to sequential processing. However, a potential limitation lies in the efficiency of the ILP solver, which may pose challenges for larger sets of smart contracts. Future research directions include exploring more efficient and scalable partitioning techniques without relying solely on ILP solvers. This paper focuses on PoW and is grounded in data collected from Ethereum. However, it is important to acknowledge that Ethereum

Table 3: Speed Up Comparisons.

Reference	Latest Year Test	Availability	Evaluation Execution Environment	Cores	Speed Up Rate	
					Proposer	Attestor
Conthereum [71], 2025	2025	Public[1]	Intel Core i5-1135G7 (2.40GHz), eight logical processors, 16.0 GB RAM, Windows 11 Pro	3	2.92	2.27
				4	3.73	2.77
				5	4.31	3.23
				32 (0%conflict)	28.58	28.58
				32 (15%conflict)	9.08	5.58
[22], 2020	N/A	Not available	4-core Intel Xeon W3550 (3.07 GHz), 12 GB RAM, Ubuntu 16	3 + 1 for GC	1.33	1.69
[55], 2019	2017	Not available	N/A	16	1.13	N/A
				64	2.26	N/A
Block-STM [26], 2023	2023	public[2]	AmazonWeb Services c5a.16xlarge instance, (AMD EPYC CPU and 128GB memory) Ubuntu 18.04	32 (low-contention)	20	-
				32 (high-contention)	9	-
ScalaProust [21], 2019	-	Public[3]	Amazon EC2 m4.10xlarge instance, which has 40 vCPUs and 160 GB of RAM	-	-	-
[68], 2023	N/A	Private	4-core processor of 2.2 MHz, 4 GB main memory	4	3	N/A
				20 to 24	5	N/A

[1] https://github.com/Conthereum,
[2] https://github.com/danielxiangzl/Block-STM,
[3] https://github.com/ScalaProust/ScalaProust

transitioned from PoW to PoS during "the merge" in September 2022. Therefore, this study conducted in 2017 requires re-evaluation or revision to align with the changes resulting from Ethereum's transition to PoS.

5.3.4 Intra-block Performance Comparison

Table 3 presents a comparative analysis of the peak performance of various concurrency approaches at the intra-block level in blockchain, sorted in descending order based on the year of presentation. The `Reference` column specifies the cited work along with the year of publication. `Latest Year Test` denotes the most recent year in which blockchain transaction characteristics were evaluated within the respective study. `Evaluation Execution Environment` describes the hardware and software configurations of the experimental environment, including processor specifications, memory capacity, and operating system, as these factors can significantly impact performance results.

The `Cores` column represents the number of processor cores explicitly allocated for transaction scheduling and execution. GC in this column refers to the cores that are allocated for garbage collection (GC) or other system processes. The `Speed Up Rate` column reports the speedup achieved in each study, further categorized into two distinct metrics: `Proposer` and `Attestor`, denoted as an ordered pair (p, a) for comparative analysis. Since attestors must maintain transaction order within a block, their speedup values are inherently lower than those of proposers for any given row in the table. If a value, including attestor performance, is not reported in a study, it is denoted as `N/A` (Not Available). For studies that present results across multiple core configurations, this table includes only representative values that allow for a comprehensive comparison.

The performance trends indicate that Conthereum [71] demonstrates the highest recorded speedup, achieving (2.92, 2.27) with 3 cores, surpassing the results of [22], which reported (1.33, 1.69) for the same core count. Notably, Conthereum's proposer

speedup of 2.92 with just 3 cores exceeds the 16-core and 64-core speedups of [55], which reported 1.13 and 2.26, respectively, without attestor evaluation. Additionally, [68] did not examine attestor performance but tested proposers ranging from 4 to 24 cores, reporting a speedup of 3 for 4 cores and 5 for 24 cores. These values are outperformed by Conthereum, which achieves a proposer speedup of 3.73 for 4 cores and 5.81 for only 9 cores. This comparative analysis highlights the efficiency of Conthereum in achieving superior speedup with fewer computational resources.

5.4 Category 4: Directed Acyclic Graph (DAG) approaches

This category of solutions redefines the structural foundation of blockchain systems by replacing the traditional linear sequential chain of blocks with a *Directed Acyclic Graph (DAG)* [60]. The promise of DAG-based blockchain systems is to enable fast confirmation (complete transactions within million seconds) and high scalability (attach transactions in parallel) without significantly compromising security.

In DAG-based systems, transactions are represented as nodes, with directed edges indicating dependencies or confirmations between them. This approach eliminates the sequential bottlenecks inherent in conventional blockchains, such as Bitcoin's PoW and Ethereum's PoS, enabling parallel transaction processing and significantly enhancing scalability and concurrency. DAG allows users to validate prior transactions to confirm their own, reducing energy consumption and transaction costs. By improving throughput and reducing latency, DAG-based architectures are particularly well-suited for high-frequency or microtransaction environments.

Despite these advantages, DAG-based systems face significant challenges, such as ensuring network consistency, preventing double-spending attacks in low-activity networks, and maintaining resilience against attacks that exploit transaction dependencies. These limitations, combined with DAG's early stage of development and the absence of proven security guarantees, explain why some well-established blockchains do not use it. In particular, security is often sacrificed for speed, and the blockchain trilemma cannot be fully resolved by DAG alone.

Several well-known DAG-based systems include IOTA, which uses the Tangle where each transaction confirms two prior transactions for enhanced scalability; Nano, which employs a block-lattice structure for fast, fee-less transactions by allowing each account to maintain its own blockchain; and Hashgraph, which offers a DAG-based consensus protocol focused on fairness, security, and efficiency.

In conclusion, while DAG-based systems have become a well-established area of research in blockchain concurrency, this survey focuses on the broader landscape of concurrency in blockchain systems. Extensive literature on DAGs is covered in several key surveys, and readers are directed to those sources for in-depth analysis. This survey aims to fill critical gaps in the concurrency domain, which remains underexplored. For specialized surveys on specific concurrency aspects, we provide references, ensuring a comprehensive overview while guiding readers to relevant detailed studies. For further exploration of DAG-specific advancements, please refer to [63], [66], [65], [24], and[41].

5.5 Category 5: Semi-Centralized Solutions

Semi-centralized solutions in blockchain technology aim to balance the benefits of decentralization with the efficiency of centralized control. These approaches often seek to enhance scalability, transaction throughput, and governance by introducing certain centralized elements while maintaining the core principles of blockchain.

An illustrative example of a semi-centralized approach is the RCANE architecture [61]. RCANE proposes a network of parallel blockchains managed under a semi-centralized framework. This design allows for concurrent processing of transactions across multiple chains, effectively distributing the load and enhancing overall system performance. The semi-centralized governance model facilitates coordinated decision-making and efficient management, addressing scalability challenges inherent in fully decentralized systems.

5.6 Category 6: Off-chain solutions

This research category discussed an intrinsic restriction of the blockchain in executing complex smart contracts. This limitation is made by considering gas to execute each instruction of a smart contract (Section 3.2) and a limited threshold of gas supported for any block. For instance, sorting 256 integers using the selection sort algorithm demands 17 million gas, exceeding the current block limit of 8 million. The quick sort also reaches this gas limit after processing 2,000 elements. Consequently, more sophisticated applications, such as decentralized blockchain oracles, become infeasible to execute under these constraints [67]. These smart contracts are name *complex* [67] or Computationally Intensive Contracts (CIC) [20]. There are some solutions to increase these per-block execution limits for complex smart contracts, and one of them is using *off-chain* mechanism. An off-chain execution model, allows contract issuers to delegate contract execution to a selected group of service providers, thus operating independently from the blockchain's consensus layer. This section of the survey encompasses the concurrent solutions in this category of architecture.

ACE (Asynchronous and Concurrent Execution of Complex Smart Contracts) [67] facilitates the execution of more complex smart contracts on permissionless blockchains using off-chain and the main benefit of ACE compared to earlier solutions is its ability to let one contract securely invoke another contract, even when they are executed by different sets of service providers. The system's key innovations include a flexible trust model and a robust concurrency control protocol, making it a significant advancement over previous solutions. The evaluation results show that ACE can facilitate the development of smart contracts that are significantly more complex than those on standard Ethereum.

In this protocol, concurrency is integrated into the consensus cycle, enabling numerous nodes to generate blocks simultaneously. PoV, in contrast, is an incentive mechanism designed to reward those who create *value*. The value is defined for any smart contract based on the smart contract token transaction volume. POV allocates more tokens to the owners of smart contracts with higher transaction volumes. POV incentivizes value creators and manages the supply of coins through an adjustable incentive coefficient algorithm. To enhance POV performance on permissioned blockchains,

the authors proposed and developed *Hypernet*, a rapid off-chain transaction system that allows transactions to be processed independently of the blockchain. It consists mainly of the client and server components. The Hypernet client sends transactions through a centralized server that does not require trust. The centralized server's role in Hypernet is merely to forward tamper-resistant transaction messages. Hypernet introduces the concepts of multi-signature address and contract address to facilitate secure and fast off-chain transactions. This solution is evaluated by generating random parameters to trigger contracts and recording the actual time taken to issue rewards. Their results indicate that the system incurs a loss of approximately 2% when POV is implemented, and Hypernet achieves transaction speeds four times faster than traditional permissioned blockchain systems.

[69] introduces SlimChain, a novel blockchain system designed to enhance transaction scalability through off-chain storage and parallel processing. Emphasizing a stateless approach, SlimChain retains only short commitments of ledger states on-chain, while off-chain nodes handle transaction executions and data storage. New schemes are proposed for off-chain smart contract execution, on-chain transaction validation, and state commitment. Additionally, optimizations are included to minimize network transmissions, and a new sharding technique is introduced to further boost system scalability. Extensive experiments validate SlimChain's performance, demonstrating a reduction in on-chain storage requirements by 97% to 99% and an improvement in peak throughput by 1.4 to 15.6 times compared to existing systems.

5.7 Category 7: Cross-chain solutions

Cross-chain solutions constitute a distinct category in smart contract concurrency due to their role in enabling atomic transactions and state synchronization across isolated blockchain networks. Unlike single-chain concurrency models, these approaches must address heterogeneous consensus protocols, varying finality times, and trust-minimized bridging mechanisms-all while preserving atomicity and preventing double-spending. Foundational work by [31] established the theoretical framework for cross-chain swaps, while later surveys ([15, 27, 45, 49, 53]) systematized the landscape.

Key concurrency challenges include: i) race conditions during cross-chain contract execution due to delayed finality, ii) verification of atomicity in multi-chain transactions, and iii) composability risks when smart contracts span multiple VMs.

5.8 Category 8: Transaction Ordering Dependency(TOD) and Front-Running Attacks

This category focuses on race conditions resulting from different miners' decisions regarding transaction orders within persisted blocks. As discussed in Section 3.2, each block of the blockchain contains verified transactions. *Transaction Ordering Dependency (TOD)* refers to the uncertainty concerning the state of a contract from the user's perspective within the blockchain system at the time of contract invocation. When multiple transactions invoking the same contract occur almost simultaneously and are included in one block, the miner arbitrarily determines their order. This can lead to

uncertainty for users regarding the state of the contract at the time of their transaction's execution, depending on the miner's decision. It is important to note that not all smart contracts are sensitive to transaction orders. Consequently, contracts affected by transaction ordering are termed TOD contracts. TOD contracts are susceptible to both benign and malicious invocations, the former potentially resulting in unexpected outcomes and the latter being exploited for unfair profit or theft [42]. Malicious exploitation of TOD-vulnerable smart contracts is termed *front-running attacks*, which originate on the Chicago Board Options Exchange (CBoE) in 1988 [46]. Its definition by the Securities Exchange Commission (SEC) in 1977 is as follows: "The practice of effecting an option transaction based upon nonpublic information regarding an impending block transaction in the underlying stock, to obtain a profit when the options market adjusts to the price at which the block trades [56]". In a blockchain system, front-running occurs when malicious entities exploit the visibility of pending transactions in the mempool to execute transactions for their own benefit at the expense of the original transaction owner. In this attack adversaries can influence the outcome of this transaction race to prioritize their transaction order in the block through participation in mining, offering higher gasPrice to incentivize miners, or collusion with other miners.

As a benign scenario example, consider a smart contract that publishes puzzles and rewards participants for correct solutions. Suppose the contract owner can alter the reward amount by a contract invocation. In such cases, when a participant submits their solution transaction, there is no certainty they will receive the current prize amount if their solution is correct. This uncertainty arises because the owner may simultaneously submit a transaction to change the prize amount, and depending on the miner's decision, the participant may receive either the old or updated reward. Another example is a decentralized exchange or marketplace allowing users to buy and sell tokens, with fluctuating price functions callable by token owners. If a buy order and a price change transaction are submitted nearly simultaneously and included in the same block, the buyer cannot be certain whether they will purchase the token at the original or updated price, depending on the transaction order.

In malicious scenarios, the owner of a puzzle smart contract could monitor the network and, upon observing a solution being submitted, quickly send a transaction to change the reward amount to zero within the block interval (explained in 3.2), increasing the likelihood of their transaction being included before the solution transaction in the subsequent block. Consequently, the adversary can exploit the system to acquire puzzle solutions at minimal cost.

Detecting TOD involves recognizing valid transactions that influence a contract's global or state variables, akin to identifying read-after-write dependencies in race detection, which poses challenges for developers. The study conducted in [47] identifies different types of TODs, including a previously undocumented variant. To address TOD detection, the paper proposes TODler, a static analyzer based on information flow analysis. Evaluation of 108 Ethereum smart contracts demonstrates that TODler surpasses existing methods in terms of both speed and accuracy, while also identifying the newly discovered TOD pattern.

In [44] the authors introduce TransRacer, an innovative tool designed to detect transaction races in Ethereum smart contracts. Transaction races, which can result from the concurrent execution of transactions within the same block, have been largely

overlooked despite their potential to cause inconsistent states and other unexpected outcomes. TransRacer employs symbolic execution to analyze function dependencies, enabling it to identify and generate witness transactions that reveal hidden races in specific contract states. This method significantly narrows the search space compared to random fuzzing techniques, enhancing detection accuracy and efficiency. Experimental results on 50 real-world smart contracts demonstrated TransRacer's capability to detect 426 races within approximately 256 minutes, including 149 significant race bugs. Further empirical analysis on 6,943 smart contracts underscores the prevalence and potential harm of transaction races in practice, highlighting the critical need for tools like TransRacer to maintain smart contract integrity.

[58] presents an open-source simulation tool designed to educate users on the vulnerabilities and mitigations associated with front-running attacks in Ethereum. This work introduces a comprehensive simulation environment that allows users to perform and understand various types of front-running attacks, such as displacement attacks, sandwich attacks, and priority gas auctions. The tool also demonstrates how to mitigate these attacks using the MEV-geth protocol, enhancing transaction privacy. By providing a hands-on educational platform, the simulation software aims to bridge the gap in educational resources regarding blockchain security, particularly in teaching smart contract vulnerabilities and mitigation strategies. This tool has been integrated into the curriculum at the University of Bristol, underscoring its utility in training the next generation of blockchain developers. The simulation's experimental framework highlights the practical challenges and potential solutions in managing front-running attacks, contributing valuable insights into the security dynamics of blockchain transactions.

5.9 Category 9: Available concurrent transaction execution vulnerabilities (misconception)

In 2017, [57] introduced a concurrent perspective on smart contracts. It explained the similarities between the behaviors of smart contracts within cryptocurrency frameworks and the classic challenges encountered in shared-memory concurrency scenarios. By explaining two instances sourced from the Ethereum blockchain, the authors explained vulnerabilities to bugs that mirror those commonly encountered in traditional concurrent programs. In elaborating on these examples, the paper underscores the intrinsic relationship between observable contract behaviors and well-established concepts in concurrency, including atomicity, interference, synchronization, and resource ownership.

This study analogized smart contract utilization in blockchains to threads that engage with concurrent objects in shared memory (*contracts-as-concurrent-objects*). Notably, the paper critiques the prevalent non-modular design of locking contracts, advocating instead for the implementation of synchronization primitives as standalone libraries. This proposed shift aligns with established software engineering best practices, where such modularization fosters better reasoning about contract behaviors. However, the separation of contract logic from locking mechanisms presents its own set of challenges, necessitating a holistic approach to contract verification that considers

interactions with rigorously specified external contracts.

Moreover, the discourse extends to address concerns regarding liveness properties in contract implementations involving locks and exclusive access. By contemplating scenarios where certain accounts may indefinitely retain access, impeding progress and fairness within the system, the paper raises pertinent questions about ensuring eventual progress and incentivizing parties to release locks. This discussion underscores the importance of fairness assumptions akin to those found in concurrency theory and presents avenues for leveraging existing proof methods for reasoning about progress and termination in multi-contract executions.

Concluding with reflections on the broader implications of the proposed analogy, the paper advocates the use of insights from concurrency research to enhance understanding, debugging, and verifying complex contract behaviors within distributed ledger environments. Although acknowledging the limitations and unique aspects of contract programming, such as gas-bounded executions and fund management, the authors invite speculation on potential challenges and insights inspired by the observed parallels. These speculations range from exploring scenarios reminiscent of the ABA problem in non-garbage-collected languages to considering the influence of mining protocols on scheduling priorities and defining consistency notions for composite contracts with multi-transactional operations. Through this comprehensive examination, the paper underscores the interdisciplinary nature of blockchain research and its potential to benefit from insights gleaned from established fields such as concurrency theory.

[51] discusses the vulnerability of smart contracts, with a particular focus on concurrency related issues. The study highlights the critical challenges posed by the concurrent nature of smart contracts, which share similarities with concurrent programs with shared memory. Using formal verification methods, specifically the *Communicating Sequence Processes (CSP)* theory [34] and the *Failure Divergence Refinement (FDR)* model checking tool, the research aims to address these challenges. The paper illustrates a race condition in a smart contract example borrowed from Solidity language documents, named The Safe Remote Purchase smart contract [3], alongside an attack scenario resulting from specific concurrent executions of this smart contract. The authors utilize CSP theory to formally model this smart contract and the considered attack, subsequently using the FDR model checker to demonstrate the possibility of this attack occurring. The findings underscore the potential advantages of using CSP and FDR tools for vulnerability detection within smart contracts, particularly in the context of concurrency.

However, our analysis reveals an error in the original assumption made in this paper regarding the race condition and the designed attack in smart contracts, specifically within Ethereum which is the subject of this study. The truth is that inherently, Ethereum's infrastructure does not support concurrency, contrary to the assumption made in this study. The concurrency assumption in [51] is based on the study in [57], which has led to ongoing debate despite its flaws. Consequently, it underscores the importance of rectifying such misconceptions within this branch of the body of literature in this field; the next section discusses this in more detail.

24

6   Addressing the Flawed Assumption (Category 9)

This section delves into an erroneous assumption identified in the investigation, categorized as Category 9, as elaborated in Section 5.9. The assumption is the presence of concurrent executions within smart contracts, implying the possibility of race conditions in the execution of multiple transactions inside one smart contract in the Ethereum infrastructure. However, this section demonstrates why this assumption is invalid. The first subsection presents evidence from the Ethereum references to refute the notion of concurrent execution. References from Ethereum's architecture clearly outline the sequential and non-concurrent execution of transactions within each block that consequently prevent any concurrent executions inside one smart contract. The subsequent subsection explores an alternative perspective on the potential for concurrent execution through the mechanism of sharding.

6.1   Evidence from blockchain design

Ethereum. Buterin et al. explicitly stated the sequential execution of transactions on Ethereum white paper as "Note that the state is not encoded in the block in any way; it is purely an abstraction to be remembered by the validating node and can only be computed (securely) for any block starting from the genesis state and sequentially applying every transaction in every block" [16].

6.2   Sharding and potential for transactions concurrency issue

Sharding is a scalability technique that partitions the blockchain state and transaction processing across multiple shards (databases). This allows for parallel processing of transactions, potentially enabling true concurrency in smart contracts.

6.2.1   Sharding architecture

In a sharded architecture, transactions are routed to their designated shard based on a sharding key (e.g., user address). This enables concurrent processing of transactions within different shards. However, challenges remain:

- **Cross-Shard Transactions:** Transactions that interact with data in multiple shards require careful coordination to maintain consistency.

- **State Synchronization:** Sharding introduces the need for efficient mechanisms to keep all shard states consistent with the overall blockchain state.

- **Security Considerations:** Careful design is necessary to ensure the security of the sharded system and prevent attacks that exploit shard boundaries.

6.2.2   Potential benefits of sharding for concurrency

Despite the challenges, sharding offers potential benefits for achieving true concurrency in smart contracts:

Table 4: Comparison

Year	Ref	Miner Approach	Locks	Require Block Graph	Validator Approach	Blockchain Type
2020	Dickerson et al. [22]	Pessimistic ScalaSTM	Yes	Yes	Fork-join	Permissionless
2018	Zhang and Zhang [72]	-	-	Read, Write Set	MVTO Approach	Permissionless
2019	Anjana et al. [7]	Optimistic RWSTM	No	Yes	Decentralized	Permissionless
2019	Amiri et al. [5]	Static Analysis	-	Yes	-	Permissioned
2019	Saraph and Herlihy [55]	Bin-based Approach	Yes	No	Bin-based	Permissionless
2019	Anjana et al. [8]					
2021	Anjana et al. [9]	Optimistic ObjectSTM	No	Yes	Decentralized	Permissionless
2021	Anjana [6]					
2022	Baheti et al. [12]					
2023	Piduguralla et al. [50]					
2024	Anjana et al. [10]	Bin+Optimistic RWSTM	No	No (if no dependencies)/Yes	Decentralized	Permissionless

- **Increased Throughput:** Parallel processing can significantly increase the number of transactions processed per second.

- **Improved Scalability:** Sharding can accommodate a growing number of users and transactions.

- **Enhanced Flexibility:** Developers can design smart contracts that take advantage of shard-specific functionalities.

7 Conclusion and future work

In this paper, we have critically examined the assumptions surrounding concurrency in smart contracts, identifying misconceptions in the discourse on concurrent transaction execution. Through a comprehensive analysis of existing literature and concepts such as sharding, we have challenged prevailing paradigms and highlighted the need for a nuanced understanding of concurrency in blockchain systems. Moving forward, we advocate for continued research and development efforts to address the inherent trade-offs between scalability, security, and decentralization in blockchain networks.

Acknowledgments

References

[1] Proof-of-stake (pos), 2024. URL https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

[2] Blockchain.com | charts - blockchain size (mb), 2024. URL https://www.blockchain.com/explorer/charts/[id].

[3] Solidity by example âĂŤ solidity 0.8.26 documentation, 2024. URL https://docs.soliditylang.org/en/latest/solidity-by-example.html# safe-remote-purchase.

[4] Firas Hammoodi Neamah Al-Mutar, Ahmed Ali Talib Al-Khazaali, and Baqar Assam Hataf. Scalability of blockchain: Review of cross-sharding with high communication overhead. In *BIO Web of Conferences*, volume 97, page 00075. EDP Sciences, 2024.

[5] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. Par-blockchain: Leveraging transaction parallelism in permissioned blockchain systems. In *Proceedings - International Conference on Distributed Computing Systems*, volume 2019-July, pages 1337–1347, 2019. doi: 10.1109/ICDCS.2019.00134. URL https://www.scopus.com/inward/ record.uri?eid=2-s2.0-85074865320&doi=10.1109%2fICDCS.2019. 00134&partnerID=40&md5=da89308a8d5454664d2fe873b8e8009d. Cited by: 60; All Open Access, Green Open Access.

[6] Parwat Singh Anjana. Efficient parallel execution of block transactions in blockchain. In *Middleware 2021 Doctoral Symposium - Proceedings of the 22nd International Middleware Conference: Doctoral Symposium*, pages 8–11, 2021. doi: 10.1145/3491087.3493676. URL https://www.scopus.com/ inward/record.uri?eid=2-s2.0-85121106502&doi=10.1145%2f3491087. 3493676&partnerID=40&md5=58bbd83f3256405ca871f41dbda52aa8. Cited by: 3.

[7] Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. An efficient framework for optimistic concurrent execution of smart contracts. In *Proceedings - 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2019*, pages 83–92, 2019. doi: 10.1109/EMPDP.2019.8671637. URL https://www.scopus.com/inward/ record.uri?eid=2-s2.0-85063875625&doi=10.1109%2fEMPDP.2019. 8671637&partnerID=40&md5=d739ee512c8d47e1f9ab15356e94d63d. Cited by: 45; All Open Access, Green Open Access.

[8] Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. Entitling concurrency to smart contracts using optimistic transactional memory. In *ACM International Conference Proceeding Series*, page 508, 2019. doi: 10.1145/3288599.3299723. URL https://www.scopus.com/ inward/record.uri?eid=2-s2.0-85060934158&doi=10.1145%2f3288599. 3299723&partnerID=40&md5=7993743a21a60f85c7159688d1533d58. Cited by: 2.

[9] Parwat Singh Anjana, Hagit Attiya, Sweta Kumari, Sathya Peri, and Archit Somani. Efficient concurrent execution of smart contracts in blockchains using object-based transactional memory. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12129 LNCS:77–93, 2021. doi: 10.1007/978-3-030-67087-0_6.

URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85101539609&doi=10.1007%2f978-3-030-67087-0_6&partnerID=40&md5=445f0bc290b7bfef3c7c0c869e9d3b0d. Cited by: 6; All Open Access, Green Open Access.

[10] Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. Optsmart: a space efficient optimistic concurrent execution of smart contracts. *Distributed and Parallel Databases*, 42(2):245–297, 2024. doi: 10.1007/s10619-022-07412-y. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85129749817&doi=10.1007%2fs10619-022-07412-y&partnerID=40&md5=fe84cbbb3450d096d6079ad0914c7296. Cited by: 0; All Open Access, Green Open Access.

[11] Basem Assiri and Wazir Zada Khan. Fair and trustworthy: Lock-free enhanced tendermint blockchain algorithm. *Telkomnika (Telecommunication Computing Electronics and Control)*, 18(4):2224 – 2234, 2020. doi: 10.12928/TELKOMNIKA.V18I4.15701. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087566813&doi=10.12928%2fTELKOMNIKA.V18I4.15701&partnerID=40&md5=a3f2ded577d7ecb8b53fd7a5b85f2738. Cited by: 6; All Open Access, Green Open Access, Hybrid Gold Open Access.

[12] Shrey Baheti, Parwat Singh Anjana, Sathya Peri, and Yogesh Simmhan. Dipetrans: A framework for distributed parallel execution of transactions of blocks in blockchains. *Concurrency and Computation: Practice and Experience*, 34(10), 2022. doi: 10.1002/cpe.6804. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85122252293&doi=10.1002%2fcpe.6804&partnerID=40&md5=7c971a4acf78188851ee429342ff56e5. Cited by: 9; All Open Access, Green Open Access.

[13] Yongjie Bai, Yang Zhi, Hui Li, Han Wang, Ping Lu, and Chengtao Ma. On parallel mechanism of consortium blockchain: Take pov as an example. In *ACM International Conference Proceeding Series*, pages 147 – 154, 2021. doi: 10.1145/3460537.3460560. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85115674506&doi=10.1145%2f3460537.3460560&partnerID=40&md5=f01c2263d7c6e4f9d9a2a06584ab07cb. Cited by: 3.

[14] Md Mohaimin Al Barat, Shaoyu Li, Changlai Du, Y. Thomas Hou, and Wenjing Lou. Sok: Public blockchain sharding. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 766–783, 2024. doi: 10.1109/ICBC59979.2024.10634422.

[15] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *ACM Comput. Surv.*, 54(8), October 2021. ISSN 0360-0300. doi: 10.1145/3471140. URL https://doi-org.ezp.biblio.unitn.it/10.1145/3471140.

[16] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013. URL https://github.com/ethereum/wiki/wiki/White-Paper.

[17] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[18] Long Chen, Lin William Cong, and Yizhou Xiao. *A Brief Introduction to Blockchain Economics*, chapter Chapter 1, pages 1–40. worldscientific, 2021. doi: 10.1142/9789811220470_0001. URL https://www.worldscientific.com/doi/abs/10.1142/9789811220470__0001.

[19] Mauro Conti, Ankit Gangwal, and Michele Todero. Blockchain trilemma solver algorand has dilemma over undecidable messages. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450371643. doi: 10.1145/3339252.3339255. URL https://doi.org/10.1145/3339252.3339255.

[20] Sourav Das, Vinay Joseph Ribeiro, and Abhijeet Anand. Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes, 2018. URL https://arxiv.org/abs/1811.03265.

[21] Thomas Dickerson, Eric Koskinen, Paul Gazzillo, and Maurice Herlihy. Conflict abstractions and shadow speculation for optimistic transactional objects. In Anthony Widjaja Lin, editor, *Programming Languages and Systems*, pages 313–331, Cham, 2019. Springer International Publishing.

[22] Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. Adding concurrency to smart contracts. *Distributed Computing*, 33(3-4):209–225, 2020. doi: 10.1007/s00446-019-00357-z. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85068876264&doi=10.1007%2fs00446-019-00357-z&partnerID=40&md5=dd7311fa1468b935494b5c6719d86e3e. Cited by: 29; All Open Access, Green Open Access.

[23] etherscan.io. Ethereum full node sync (archive) chart | etherscan, 2024. URL https://etherscan.io/chartsync/chainarchive.

[24] Xiang Fu, Huaimin Wang, and Peichang Shi. A survey of blockchain consensus algorithms: mechanism, design and applications. *Science China Information Sciences*, 64:1–15, 2021.

[25] Zhimin Gao, Lei Xu, Lin Chen, Nolan Shah, Yang Lu, and Weidong Shi. Scalable blockchain based smart contract execution. In *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 352–359, 2017. doi: 10.1109/ICPADS.2017.00054.

[26] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. Block-stm: Scaling blockchain

execution by turning ordering curse to a performance blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, page 232âĂŞ244, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700156. doi: 10.1145/ 3572848.3577524. URL https://doi.org/10.1145/3572848.3577524.

[27] Panpan Han, Zheng Yan, Wenxiu Ding, Shufan Fei, and Zhiguo Wan. A survey on cross-chain technologies. *Distrib. Ledger Technol.*, 2(2), June 2023. doi: 10. 1145/3573896. URL https://doi-org.ezp.biblio.unitn.it/10.1145/3573896.

[28] Faiza Hashim, Khaled Shuaib, and Nazar Zaki. Sharding for scalable blockchain networks. *SN Computer Science*, 4(1):2, 2022.

[29] Shihab Shahriar Hazari and Qusay H. Mahmoud. A parallel proof of work to improve transaction speed and scalability in blockchain systems. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019*, pages 916–921, 2019. doi: 10.1109/CCWC.2019.8666535. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85063863999& doi=10.1109%2fCCWC.2019.8666535&partnerID=40&md5= 48e48f90293f465ef556ece7331b6ae9. Cited by: 69.

[30] Shihab Shahriar Hazari and Qusay H. Mahmoud. Improving transaction speed and scalability of blockchain systems via parallel proof of work. *Future Internet*, 12(8), 2020. doi: 10.3390/FI12080125. URL https://www.scopus.com/inward/record.uri?eid=2-s2. 0-85089548009&doi=10.3390%2fFI12080125&partnerID=40&md5= ae0d6542c52eb6252771b3998d65a3dc. Cited by: 34; All Open Access, Gold Open Access.

[31] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, page 245âĂŞ254, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357951. doi: 10.1145/3212734.3212736. URL https: //doi-org.ezp.biblio.unitn.it/10.1145/3212734.3212736.

[32] Maurice Herlihy and Eric Koskinen. Transactional boosting: a methodology for highly-concurrent transactional objects. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '08, pages 207–216, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595937957. doi: 10.1145/1345206.1345237. URL https://doi.org/10.1145/1345206.1345237.

[33] Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer. Software transactional memory for dynamic-sized data structures. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing*, PODC '03, pages 92–101, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137087. doi: 10.1145/872035.872048. URL https://doi.org/10.1145/872035.872048.

30

[34] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8): 666–677, aug 1978. ISSN 0001-0782. doi: 10.1145/359576.359585. URL https://doi.org/10.1145/359576.359585.

[35] Gang Huang, Kaidong Wu, Chaoran Luo, Su Zhang, Huaqian Cai, Xiang Jing, and Yun Ma. Bdledger: A scalable distributed ledger for large-scale data recording. *Communications in Computer and Information Science*, 1490 CCIS:87–100, 2021. doi: 10.1007/978-981-16-7993-3_7. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85121764911&doi=10.1007%2f978-981-16-7993-3_7&partnerID=40&md5=5d89292da80717632830c135dcd1fe9c. Cited by: 0.

[36] Jae Kwon. tendermint/tendermint. URL https://github.com/tendermint/tendermint. original-date: 2014-05-14T23:21:35Z.

[37] Bahareh Lashkari and Petr Musilek. A comprehensive review of blockchain consensus mechanisms. *IEEE Access*, 9:43620–43652, 2021. doi: 10.1109/ACCESS.2021.3065880.

[38] Kejiao Li, Hui Li, Hanxu Hou, Kedan Li, and Yongle Chen. Proof of vote: A high-performance consensus protocol based on vote mechanism & consortium blockchain. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 466–473, 2017. doi: 10.1109/HPCC-SmartCity-DSS.2017.61.

[39] Jian Liu, Peilun Li, Raymond Cheng, N. Asokan, and Dawn Song. Parallel and asynchronous smart contract execution. *IEEE Transactions on Parallel and Distributed Systems*, 33(5):1097–1108, 2022. doi: 10.1109/TPDS.2021.3095234. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85112672240&doi=10.1109%2fTPDS.2021.3095234&partnerID=40&md5=9b639907ed892ed22587c0ec40947605. Cited by: 12; All Open Access, Green Open Access.

[40] Xinmeng Liu, Haomeng Xie, Zheng Yan, and Xueqin Liang. A survey on blockchain sharding. *ISA Transactions*, 141:30–43, 2023. doi: 10.1016/j.isatra.2023.06.029. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85164573433&doi=10.1016%2fj.isatra.2023.06.029&partnerID=40&md5=1f360f22b8f3ad34707b24e44bde5a6c. Cited by: 1.

[41] Xiaofeng Lu, Cheng Jiang, and Pan Wang. A survey on consensus algorithms of blockchain based on dag. In *Proceedings of the 2024 6th Blockchain and Internet of Things Conference*, BIOTC '24, pages 50–58, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400717000. doi: 10.1145/3688225.3688232. URL https://doi-org.ezp.biblio.unitn.it/10.1145/3688225.3688232.

[42] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. CCS '16, pages 254–269, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi: 10.1145/2976749.2978309. URL https://doi.org/10.1145/2976749.2978309.

[43] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi: 10.1145/2976749.2978389. URL https://doi-org.ezp.biblio.unitn.it/10.1145/2976749.2978389.

[44] Chenyang Ma, Wei Song, and Jeff Huang. Transracer: Function dependence-guided transaction race detection for smart contracts. In *ESEC/FSE 2023 - Proceedings of the 31st ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 947 – 959, 2023. doi: 10.1145/3611643.3616281. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85180548834&doi=10.1145%2f3611643.3616281&partnerID=40&md5=250a0fdf5719e743a09df50aa0cb43e6. Cited by: 0.

[45] Hanyu Mao, Tiezheng Nie, Hao Sun, Derong Shen, and Ge Yu. A survey on cross-chain technology: Challenges, development, and prospect. *IEEE Access*, 11:45527–45546, 2023. doi: 10.1109/ACCESS.2022.3228535.

[46] Jerry W Markham. Front-running-insider trading under the commodity exchange act. *Cath. UL Rev.*, 38:69, 1988.

[47] Sundas Munir and Christoph Reichenbach. Todler: A transaction ordering dependency analyzer - for ethereum smart contracts. In *Proceedings - 2023 IEEE/ACM 6th International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB 2023*, pages 9–16, 2023. doi: 10.1109/WETSEB59161.2023.00007. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85169085075&doi=10.1109%2fWETSEB59161.2023.00007&partnerID=40&md5=cc67ae8bf141b5a2f8c8a706660b3b67. Cited by: 1.

[48] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[49] Wei Ou, Shiying Huang, Jingjing Zheng, Qionglu Zhang, Guang Zeng, and Wenbao Han. An overview on cross-chain: Mechanism, platforms, challenges and advances. *Computer Networks*, 218:109378, 2022. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2022.109378. URL https://www.sciencedirect.com/science/article/pii/S1389128622004121.

[50] Manaswini Piduguralla, Saheli Chakraborty, Parwat Singh Anjana, and Sathya Peri. Dag-based efficient parallel scheduler for blockchains: Hyperledger sawtooth as a case study. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*,

14100 LNCS:184–198, 2023. doi: 10.1007/978-3-031-39698-4_13. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85171582845& doi=10.1007%2f978-3-031-39698-4__13&partnerID=40&md5= ffc244a7fab78042dfe584fc0f9d18d4. Cited by: 0.

[51] Meixun Qu, Xin Huang, Xu Chen, Yi Wang, Xiaofeng Ma, and Dawei Liu. Formal verification of smart contracts from the perspective of concurrency. In Meikang Qiu, editor, *Smart Blockchain*, pages 32–43, Cham, 2018. Springer International Publishing. ISBN 978-3-030-05764-0.

[52] Brandon Liew Yi Quan, Nur Haliza Abdul Wahab, Arafat Al-Dhaqm, Ahmad Al-shammari, Ali Aqarni, Shukor Abd Razak, and Koh Tieng Wei. Recent advances in sharding techniques for scalable blockchain networks: A review. *IEEE Access*, pages 1–1, 2024. doi: 10.1109/ACCESS.2024.3523256.

[53] Peter Robinson. Survey of crosschain communications protocols. *Computer Networks*, 200:108488, 2021. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2021.108488. URL https://www.sciencedirect.com/science/article/pii/S1389128621004321.

[54] Anthony Roscoe. The theory and practice of concurrency. 1998.

[55] Vikram Saraph and Maurice Herlihy. An empirical study of speculative concurrency in ethereum smart contracts. *arXiv preprint arXiv:1901.01376*, 2019.

[56] États-Unis. Securities and Exchange Commission. Special Study of the Options Markets. *Report of the Special Study of the Options Markets to the Securities and Exchange Commission*. US Government Printing Office, 1979.

[57] Ilya Sergey and Aquinas Hobor. A concurrent perspective on smart contracts. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security*, pages 478–493, Cham, 2017. Springer International Publishing. ISBN 978-3-319-70278-0.

[58] Zachary Stucke, Theodoros Constantinides, and John Cartlidge. Simulation of front-running attacks and privacy mitigations in ethereum blockchain. In *European Modeling and Simulation Symposium, EMSS*, 2022. doi: 10.46354/i3m.2022.emss.041. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85142909699&doi=10.46354%2fi3m.2022.emss.041&partnerID=40&md5=e02a26112e1e8e8677bb705ab0165988. Cited by: 1; All Open Access, Green Open Access.

[59] Scopus team. Scopus content | elsevier, 2024. URL https://www.elsevier.com/products/scopus/content.

[60] K. Thulasiraman and M. N. S. Swamy. *Graphs: Theory and Algorithms*. John Wiley & Sons. ISBN 978-1-118-03025-7. Google-Books-ID: rFH7eQffQNkC.

[61] Nguyen Van Toan, Ung Park, and Geunwoong Ryu. Rcane: Semi-centralized network of parallel blockchain and apos. In *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, volume 2018-December, pages 695–700, 2018. doi: 10.1109/PADSW.2018.8644573. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85063318541& doi=10.1109%2fPADSW.2018.8644573&partnerID=40&md5= 34fd356f01861d9ec6366861441a54f3. Cited by: 6.

[62] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, AFT '19, pages 41–61, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367325. doi: 10.1145/3318041.3355457. URL https://doi-org.ezp.biblio.unitn.it/10.1145/3318041.3355457.

[63] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. Sok: Dag-based blockchain systems. *ACM Comput. Surv.*, 55(12), March 2023. ISSN 0360-0300. doi: 10.1145/3576899. URL https://doi-org.ezp.biblio.unitn.it/10.1145/3576899.

[64] Yiyuan Wang, Shaowei Cai, Shiwei Pan, Ximing Li, and Monghao Yin. Reduction and local search for weighted graph coloring problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2433–2441, Apr. 2020. doi: 10.1609/aaai.v34i03.5624. URL https://ojs.aaai.org/index.php/AAAI/article/view/5624.

[65] Qian Wei, Bingzhe Li, Wanli Chang, Zhiping Jia, Zhaoyan Shen, and Zili Shao. A survey of blockchain data management systems. *ACM Trans. Embed. Comput. Syst.*, 21(3), May 2022. ISSN 1539-9087. doi: 10.1145/3502741. URL https://doi-org.ezp.biblio.unitn.it/10.1145/3502741.

[66] Huan Yu Wu, Xin Yang, Chentao Yue, Hye-Young Paik, and Salil S. Kanhere. Chain or dag? underlying data structures, architectures, topologies and consensus in distributed ledger technology: A review, taxonomy and research issues. *Journal of Systems Architecture*, 131:102720, 2022. ISSN 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2022.102720. URL https://www.sciencedirect.com/science/article/pii/S1383762122002077.

[67] Karl WÃijst, Sinisa Matetic, Silvan Egli, Kari Kostiainen, and Srdjan Capkun. Ace: Asynchronous and concurrent execution of complex smart contracts. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 587–600, 2020. doi: 10.1145/3372297.3417243. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85096161986&doi=10.1145%2f3372297.3417243& partnerID=40&md5=86515ecf98b7c301c895e45e27353ba4. Cited by: 29; All Open Access, Bronze Open Access.

[68] Huahui Xia, Jinchuan Chen, Nabo Ma, Jia Huang, and Xiaoyong Du. Efficient execution of blockchain transactions through deterministic concurrency

control. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 13943 LNCS:509–518, 2023. doi: 10.1007/978-3-031-30637-2_33. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85161654879& doi=10.1007%2f978-3-031-30637-2_33&partnerID=40&md5= 7d0d01cd2a6bb0d750eaa00ae34bc744. Cited by: 1.

[69] Cheng Xu, Ce Zhang, Jianliang Xu, and Jian Pei. Slimchain: Scaling blockchain transactions through off-chain storage and parallel processing. *Proceedings of the VLDB Endowment*, 14(11):2314–2326, 2021. doi: 10.14778/3476249.3476283. URL https://www.scopus.com/ inward/record.uri?eid=2-s2.0-85119678964&doi=10.14778%2f3476249. 3476283&partnerID=40&md5=0083c4152ed2912bdf744a0f96bcbb98. Cited by: 52.

[70] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J. Andrew Zhang, and Ren Ping Liu. Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181, 2020. doi: 10.1109/ACCESS.2020.2965147.

[71] Atefeh Zareh Chahoki, Maurice Herlihy, and Marco Roveri. Conthereum: Concurrent ethereum optimized transaction scheduling for multi-core execution. In *The 37st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '25, pages 349–358, New York, NY, USA, 2025. Association for Computing Machinery. ISBN -. doi: -. URL -.

[72] An Zhang and Kunlong Zhang. Enabling concurrency on smart contracts using multiversion ordering. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10988 LNCS:425–439, 2018. doi: 10.1007/978-3-319-96893-3_32. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85051129282& doi=10.1007%2f978-3-319-96893-3_32&partnerID=40&md5= ad208a90b3e81dd28f09f5e7c643bd57. Cited by: 21.

[73] Kaiwen Zhang and Hans-Arno Jacobsen. Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In *ICDCS*, pages 1337–1346, 2018.

Table 5: Abbreviations

AUs	Atomic Units
BFT	Byzantine Fault-Tolerant
BG	Block Graph
CIC	Computationally Intensive Contracts
CSP	Communicating Sequence Processes
DAG	Directed Acyclic Graph
DPoS	Delegated Proof of Stake
DVC	deterministic concurrency control
ET	Energy Trading
EVM	Ethereum Virtual Machine
FDR	Failure Divergence Refinement
ILP	Integer Linear Programming
Mempool	Memory Pool
MEV	Maximal Extractable Value
P2P	Peer to Peer
PBFT	Practical Byzantine Fault Tolerance
PoA	Proof of Authority
PoS	Proof of Stake
PoV	Proof of Value
PoV	Proof of Vote
PoW	Proof of Work
PPoV	Parallel Proof of Vote
SC	Smart Contract
SMuLC	Sequential Multi-Layer Consensus
STM	Software Transactional Memory
TOD	Transaction Ordering Dependency
TPS	Transaction Per Second
TTP	Trusted Third Party

Abbreviations The following abbreviations are used in this manuscript:

8   About the Authors

Atefeh Zareh Chahoki is a Ph.D. candidate in Computer Science at the University of Trento. Her research interests include blockchain system reliability, formal verification of smart contracts, and transaction scheduling optimization in distributed ledgers. She publishes in venues such as *IEEE Transactions on Information Forensics and Security* and is committed to open science-releasing all implementations as open-source software. Before commencing her doctoral studies, she acquired extensive industry experience in enterprise software development, project management, and university lecturing.

Maurice Herlihy has an A.B. in Mathematics from Harvard University, and a Ph.D. in Computer Science from M.I.T. He has served on the faculty of Carnegie Mellon University and the staff of DEC Cambridge Research Lab. He is the recipient of the 2003 Dijkstra Prize in Distributed Computing, the 2004 Gödel Prize in theoretical computer science, the 2008 ISCA influential paper award, the 2012 Edsger W. Dijkstra Prize, and the 2013 Wallace McDowell award. He received a 2012 Fulbright Distinguished Chair in the Natural Sciences and Engineering Lecturing Fellowship, and he is fellow of the ACM, the National Academy of Inventors, the National Academy of Engineering, and the National Academy of Arts and Sciences. In 2022, he won his third Dijkstra Prize.

Marco Roveri is an Associate Professor in the Department of Information Engineering and Computer Science at the University of Trento, Italy. He was a Senior Researcher in the Embedded Systems Unit of Fondazione Bruno Kessler, and before a researcher in the Automated Reasoning Division of the Istituto Trentino di Cultura. His research interests include formal verification of hardware and software systems, formal cyber security, formal requirements validation, and automated model-based planning, and application of such techniques in industrial settings.