Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (36^{th}cycle)

Blockchain and IoT Applications for Supply Chain and Transportation

By

Vittorio Capocasale

******

Supervisor(s):
Prof. Guido Perboli

Doctoral Examination Committee:
Prof. A.B. , Referee, University of...
Prof. C.D, Referee, University of...
Prof. E.F, University of...
Prof. G.H, University of...
Prof. I.J, University of...

Politecnico di Torino
2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Vittorio Capocasale
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

To my parents, a worthless token of appreciation for all they did, all they sacrificed, all they endured.

Acknowledgements

I cannot start this acknowledgemnts section without thanking my supervisor, Professor Guido Perboli, for all the time, money, and effort he invested on me for more than five years, from my Master's thesis onward. I also want to thank all the members, past and new, of his research team, of which I had the pleasure of being a member.

I want to express my sincere gratitude to Professor Fernando Pedone, who agreed to host me in his research team during my visiting period in Switzerland and keeps supporting and guiding me to this day, along with all the members of his research team, who made me feel the new place like home.

I will never be able to thank enough my familiy and friends, that keep supporting me no matter how little I deserve it. To them, I owe it all.

Finally, a warm thank you to all those who I have not explicitely mentioned here, but that, big or small, were part of this journey. Some of them, I now proudly call friends.

Abstract

Blockchain technology may solve current issues in supply chains related to the lack of timely, correct, standardized, authentic, accessible, and verifiable information. However, transitioning toward decentralized paradigms, as blockchain imposes, is not easy for companies due to strategic, managerial, technological, and environmental barriers. To overcome such barriers and support successful blockchain adoption, we discuss the development of a use case based on an electric vehicle supply chain and draw general insights from a technological standpoint, offering a practical guide for building blockchain-based applications. In particular, we propose a decision-making framework for blockchain suitability assessment; we present a methodology for comparing the performances of the various blockchain frameworks fairly; we analyze deterministic and parallel transaction execution in blockchain frameworks to enhance performance; we propose guidelines for smart contract standardization; and we discuss mitigating the "garbage in, garbage out" problem introduced by oracles in blockchain systems. Based on our results, we conclude that, with opportune design, technological barriers to blockchain adoption may be overcome, but this is insufficient to push companies to embrace decentralized paradigms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Supply chains, the intricate networks that drive global commerce, currently face challenges such as the lack of timely, correct, standardized, authentic, accessible, and verifiable information. These issues result in inefficiencies in handling paperwork and unnecessary litigation costs. Fortunately, recent technological advancements have sparked a revolution in supply chains based on data sharing and digitalization, known as Logistics 4.0. At the core of this revolution are peer-to-peer technologies like blockchain and the interplanetary file system, poised to transform the way data is shared and managed across supply chain ecosystems. Blockchain, in particular, holds the potential to address many of the current information issues in supply chains.

Blockchain is a decentralized database where changes are approved through majority-based voting. As a result, data manipulation is not possible as long as the voting majority is honest. This capability positions blockchain as a solution for safe data sharing and tamper-resistant storage among supply chain companies, potentially becoming the single source of truth for supply chain events and responsibilities.

However, the path toward blockchain adoption in supply chains is not straightforward. On the strategic side, it demands the abandonment of existing business models for decentralized ones, necessitating a disruptive paradigm shift in how value is perceived and created. On the managerial side, new practices are required for greater collaboration. However, existing misconceptions are impeding the general understanding of the benefits the technology provides and the compromises it entails. On the technical side, blockchain introduces numerous challenges, including limited efficiency, decreased confidentiality, and issues related to the trustworthiness of data

sources. Moreover, the environmental side faces challenges due to the lack of clear standards and regulations.

Unsurprisingly, many blockchain-based projects for supply chains fail to achieve tangible benefits and are quickly abandoned. The average discontinuation rate is approximately one year, with a survival rate of less than 10% [206]. Notably, Tradelens, one of the first and most recognized supply chain management platforms, recently faced termination due to its failure to achieve global industry collaboration, despite being developed by IBM, a blockchain pioneer, and Maersk, a world-leading shipping company [74].

To address the previously mentioned issues and support successful blockchain adoption, this thesis delves into the nexus of supply chain dynamics, blockchain technology, and digitalization, with a particular focus on an electric vehicle supply chain as a representative use case. Our exploration navigates the challenges and opportunities at the intersection of these domains, aiming to chart a course toward a more transparent, efficient, and collaborative future for supply chain operations. With a strong technolgical focus, we discuss the development of our use case and draw general insights, serving as a practical guide for building blockchain-based applications.

The remainder of this thesis is articulated in the following chapters.

- Chapter 2 provides some general information on blockchain technology and contextualizes this thesis within the existing literature.
- Chapter 3 describes a decision-making framework for blockchain suitability assessment and introduces a logistic use case that will be developed throughout this thesis.
- Chapter 4 provides a comparison of some of the most used permissioned blockchain frameworks. This chapter also introduces a methodology for comparing the performances of the various frameworks fairly.
- Chapter 5 discusses the possibility of leveraging parallel transaction execution for the design of our solution. To this extent, the chapter introduces the “ambiguous state representation problem” and describes an optimistic algorithm that guarantees determinism without completely sacrificing parallelization.

- Chapter 6 proposes some guidelines for smart contract standardization based on the experience we gained from designing the smart contracts for our use case.
- Chapter 7 discusses the integration of Internet of Things (IoT) devices into the blockchain-based solution for our use case. The chapter focuses on mitigating the “Garbage In, Garbage Out” (GIGO) problem introduced by oracles like IoT devices.
- Chapter 8 summarizes the main takeaways from this thesis and provides a broader outlook on blockchain adoption in logistic networks.

Chapter 2

Background

In this chapter, we provide background information on blockchain technology and the other topics that are relevant to this thesis. We also analyze the relevant literature. The contents of this chapter are based on Ref. [166, 34, 39, 27, 37, 40, 42, 38, 41].

2.1 Introduction

Logistics companies are undergoing a transformation process based on data sharing and digitalization, known as Logistics 4.0. This revolution is driven by the need to overcome well-known inefficiencies in logistics networks, particularly bureaucracy and litigations, mainly as a consequence of the inability of supply chain companies to collect and share data seamlessly and reliably.

Blockchain emerges as one of the possible solutions to this data sharing problem by offering the possibility to create a decentralized database where companies can share data without incurring the risks of unilateral or unauthorized data manipulation. However, it also introduces many new problems, which we will discuss and address (to some extent) in the next chapters of this thesis.

This chapter, instead, serves the purpose of making the reader familiar with the blockchain landscape and, to this extent, covers the following aspects.

- The description of blockchain technology and related concepts.
- The contextualization of this thesis within the existing literature.

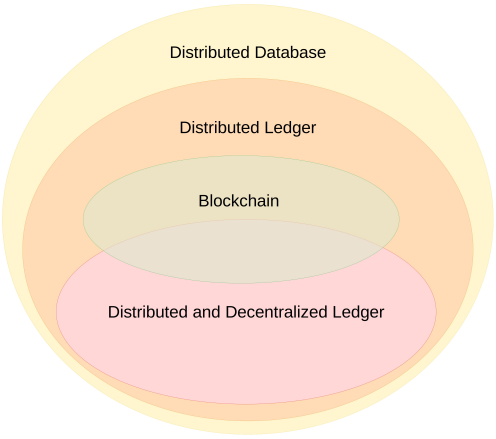


Fig. 2.1 Boundaries among distributed database, distributed ledger (DLT), and blockchain technologies.

The remainder of this chapter is articulated as follows: we introduce blockchain technology in section 2.2, contextualize this thesis in the current literature in Section 2.3, and conclude this chapter with Section 2.4.

2.2 Blockchain

In the past decade, blockchain technology has been intensively studied due to its applicability to many use cases. Nonetheless, there is no universally accepted definition for blockchain and similar technologies. Usually, all such technologies are called *distributed ledger technologies (DLTs)*, even though there may be significant differences or unclear boudaries among the various DLTs. The relashionships between distributed database, distributed ledger, and blockchain are summarized in Fig. 2.1.

DLTs are distributed databases structured as ledgers: they record the whole history of transactions (i.e., modifications) to the stored data. Each ledger exists in and multiple copies, and each copy is managed by an entity called a peer.

Blockchain is a DLT characterized by its peculiar structure of the ledger: transactions are grouped into blocks, which are added to the ledger one after the other [222]. Each block contains the hash of its predecessor, thus, a block cannot be altered without also altering all the subsequent ones. Doing so is relatively easy if a single entity manages all the copies of the ledger, but becomes nearly impossible

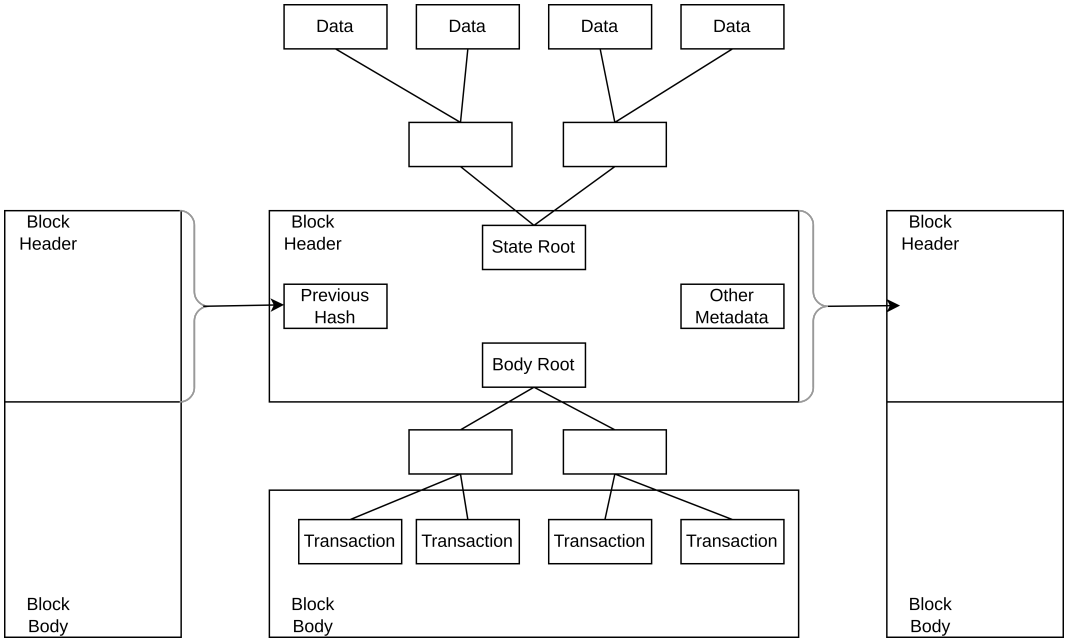


Fig. 2.2 Blockchain structure

when the ledger is managed by multiple non-trusting parties. Thus, the properties of blockchain systems depend on their governance model. This is better explained in Section 2.2.1.

A blockchain is usually composed of two databases: the history database, which is the ledger containing the sequence of transactions; and the state database, which stores the updated result of transaction execution. The state database is unnecessary, but eliminating it causes significant performance drawbacks. The state database avoids reading the whole ledger and re-executing all transactions to obtain the current data values. Fast consistency checks between the two databases at a given block height are possible by representing the state database as a Merkle tree and storing the Merkel root hash within the block. Blockchain addresses are used to access different portions of state database. The typical blockchain structure is represented in Fig. 2.2.

Blockchain works as follows [222]:

- users submit transactions to the peers. Transactions' integrity and authenticity is guaranteed through digital signatures;

- the peers agree upon which transaction to execute through consensus algorithms. The selected transactions form a block, and each peer adds it to its copy of the ledger. Simultaneously, the peers execute the transactions within the block and update their state database;
- blocks are linked through cryptographic hash functions, enabling fast data verification by only knowing the latest block hash.

2.2.1 Blockchain governance

Governance describes the power to control, coordinate and direct a blockchain system [163]. Blockchains can be categorized according to their governance model [31, 128].

- Public blockchains allow any peer to join them and participate in the consensus process. Public blockchains create trust among their participants by allowing them to obtain a full copy of the ledger and autonomously validate transactions.
- Consortium blockchains have strict rules for consensus participation and ledger interaction. Such rules are defined by the consortium members, who are the nodes participating in the consensus at a given time. Consortium blockchains can create trust among the consortium members, but external parties need to trust the consortium.
- Private blockchains are managed by a single party, which must be trusted by all the participant. Consequently, The system is centralized.

Public blockchains are permissionless because they do not pose any restrictions on participation. On the contrary, private and consortium blockchains enforce access control mechanisms and are, thus, permissioned. Nonetheless, consortium blockchains are decentrlaized while private blockchains are not, making them very different technological solutions.

2.2.2 Blockchain properties

In this section, we summarize some of the most interesting blockchain properties for the industry [222, 128, 156].

- Persistency: the ledger exists in multiple copies, making data losses unlikely.
- Decentralization: No single peer has sole custody of the ledger, except for private blockchains, which are not decentralized.
- Authenticity: digitally signatures authenticate transactions.
- Autonomy: trusted third parties are not needed to submit transactions.
- Immutability: data cannot be modified after insertion, as this would cause a mismatch between the hash of one block and the one stored in its successor. However, the whole chain of hashes can be rewritten if the majority colludes (51% attack). We underline that, in private blockchains, the managing can detains all voting power and can rewrite the chain of hashes at will.
- Auditability: peers can directly inspect their copy of the ledger. Moreover, the sequence of transactions can be re-executed at any moment to check for potential state inconsistencies.
- Resiliency: cyberattacks can be successful only if they coherently modify the majority of the copies of the ledger. Thus, decentralization is key for resiliency.
- Standardization: data encoding standards must be employed ti keep the copies identical.

2.2.3 Consensus algorithms

Blockchain peers employ consensus algorithms to keep their copies of the ledger synchronized by agreeing on the sequence of transactions to process. Various consensus algorithms can be employed, each offering different compromises in terms of efficiency and security and operating under different assumptions. Such assumptions concern the modeling of the peers (also called processes in this context) and of the network. On the processes model, Crash-Fault Tolerant (CFT) consensus algorithms only assume the presence of correct or non-responsive processes, whereas Byzantine-Fault Tolerant (BFT) consensus algorithms also deal with malicious processes that may try to disrupt the consensus. On the network model, synchronous networks are those assuming a known upper bound to message delays; asynchronous networks only require that messages are delivered in a finite (but unknown) time;

partially-synchronous networks assume the presence of a global event, called Global Stabilization Time, fired at a finite but unknown time. The network is asyncronous before GST and synchronous afterward.

In general, consensus algorithms aim to guarantee two properties:

- Safety, which is satisfied if all honest peers make the same decision, thus finding an agreement;
- Liveness, which is satisfied if a decision is made in a finite amount of time.

Nonetheless, the FLP impossibility [75] guarantees that no algorithm can satisfy both properties deterministically in the asynchronous network model. For this reason, one of the two properties can be guaranteed only probabilistically or a less general network model must be employed. If safety is guaranteed probabilistically, the consensus algorithm has probabilistic finality, which means that decisions may be reverted with a probability that decreases over time. If safety is guaranteed deterministically, the consensus algorithm has deterministic finality, which means that decisions are irreversible [15, 216].

Only BFT algorithms should be used in blockchain systems: if Byzantine nodes are absent, decisions could be delegated to a centralized system controlled by any of the nodes, making blockchain superfluous.

2.2.4 Smart contracts

Smart contracts as tamper-resistant computer programs. Smart contracts are in charge of processing transactions and their execution is replicated on each blockchain peer, guaranteeing correct results even in case a minority of the executions is faulty. Usually, smart contracts do not guarantee data confidentiality, which requires cryptographic techniques that are rarely made available in blockchain frameworks. Smart contracts can execute arbitrary logic and can automate tasks by reacting to events generated by users and other smart contracts according to pre-defined conditions. The automation of legal contracts through smart contracts is an open area of research [180].

2.2.5 Oracles

Not all data that is submitted to a blockchain system can be verified. For example, the temperature of a room provided by a sensor is only measurable by that sensor, not by the blockchain peers scattered all over the world. The providers of unverifiable data are named oracles and introduce the *Garbage In, Garbage Out* (GIGO) problem in blokcchain systems. In general, transactions passing the validity checks performed by blockchain peers may contain inaccurate data. Thus, valid does not mean correct.

2.2.6 Performance metrics

Transactions submitted to blockchain systems can be in one of the following states:

- Pending—The transaction must still appear within a block;
- Committed—The transaction has been added to a block after clearing validity checks;
- Consolidated—The transaction is permanently and irreversibly stored in the blockchain. For deterministic finality, commit time and consolidation time overlap. For probabilistic finality, transactions are consolidated after they are committed.
- Discarded—The transaction will never be added to the ledger.

Relevant metrics in blockchain systems involve read latency, transaction latency, read throughput, and transaction throughput (TPS) [103]. Read latency is the time passing between submitting a query to a node and receiving a response, transaction latency is the time required to consolidate a transaction, read throughput is the number of query-response couples a client completes in the time unit, and transaction throughput is the amount of transactions consolidated in the time unit.

2.3 Literature review

In this section, we present a summary of the studies that are relevant to this thesis. Due to the broad range of arguments we cover, we dedicate a subsection to each topic.

2.3.1 Blockchain suitability

Blockchain suitability frameworks may be divided into three categories. According to Ref. [5]: conceptual frameworks, decision models, and decision flowcharts.

In conceptual frameworks, researchers discuss the main factors to take into account based on their practical experiences. Some encompass non-technological aspects like economic or regulatory factors, whereas others focus on purely technological aspects [187, 54, 119]. In particular, Ref. [10] proposes a set of open-ended questions that should be addressed when adopting blockchain and Ref. [116] analyzes the existing literature to identify technological, organizational, and environmental enablers that help businesses in implementing blockchain and IoT solutions for secure and sustainable operations

Decision models guide blockchain adoption by leveraging mathematical constructs. For instance, BAF (Blockchain Applicability Framework) suggests the ideal blockchain solution by weighting detailed user requirements [83].

Decision flowcharts are based on graphs with nodes representing closed-ended questions and edges indicating possible answers. Users follow the path defined by their answers, culminating with an assessment on blockchain suitability. This approach is fairly common, with several multi-step frameworks for blockchain adoption emerging in the literature [161]. Some frameworks also offer implementation guidelines [21], examine blockchain-related security threats [171], and analyze real-world use cases [90, 215, 78]. Ref. [48] proposes a framework specifically designed for managers. Merging multiple frameworks into a single one is also a viable alternative [112]. Nonetheless, certain decision drivers proposed in the existing literature should be revised, in our opinion. For instance, the presence of multiple writers is an unnecessary requirement: a set of entities may want to record data written by a third party in a tamper-resistant way. In such cases, blockchain could be viable, as multiple record keepers could prevent invalid data modifications by the writer. Hence, the existence of multiple decision-makers should drive blockchain adoption, allowing keepers to determine modifiable data, instead of writers.

Ref. [132] proposes a framework of seven questions and four subquestions. However, deep technical knowledge of blockchain technology is necessary to provide accurate answers. Ref. [162] analyzes numerous factors that are overlooked in similar works and proposes a ten-step decision-making framework.

Additionally, some authors have designed decision-making frameworks for specific use cases, including the construction industry [98] and logistics [11, 77, 94]. Finally, some decision-making frameworks also suggest the most suitable blockchain platform [73]. Table 2.1 summarizes the studies analyzed in this section and compares them with the decision-making framework described in Chapter 3.

Table 2.1 Summary of decision flowchart frameworks for blockchain adoption. The table highlights which questions of our framework are also suggested by other researchers.

Ref.	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
[98]	No	Yes	No	No	No	Yes	No	No	No	No	No
[215]	No	Yes	No	No	No	No	No	No	No	No	No
[21]	No	Yes	No	No	Yes	No	No	No	No	No	No
[171]	No	Yes	No	No	Yes	No	No	No	No	Yes	Yes
[90]	No	Yes	No	No	Yes	No	No	No	No	No	No
[162]	Yes	Yes	No	No	Yes	No	Yes	No	No	Yes	No
[112]	No	Yes	No	No	No	No	No	No	No	No	No
[161]	No	Yes	No	No	No	No	No	No	No	Yes	No
[78]	No	Yes	No	No	Yes	No	No	No	No	Yes	No
[132]	Yes	Yes	No	No	Yes	No	No	No	No	Yes	No
[48]	Yes	No	No	No	No	Yes	No	Yes	No	Yes	No
Chapter 3	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.3.2 Performance and features' comparisons of blockchain frameworks

Blockchain must guarantee the same performance as other technologies to replace them in industrial information systems. For this reason, measuring the performance of blockchain frameworks is an active research field. The various studies assessing the performance of blockchain frameworks are summarized in Table 2.2. For each paper, the table details which studies analyze multiple frameworks, which present experimental performance evaluations, which adopt recent releases of the frameworks, and which attempt to reduce the differences among different frameworks to make even comparisons.

One of the earliest studies on performance evaluations of permissioned blockchain frameworks led to the creation of Blockbench [67], a performance evaluation tool for blockchains with support for various blockchain clients including Fabric v0.6.0-

Table 2.2 Summary of the studies on blockchain framework performance evaluation and comparison.

Ref.	Framework	Multiple frameworks	Experimental performance evaluation	Recent releases	Cross-framework methodology
[143]	GoQuorum v2.2.1	No	Yes	No	No
[16]	GoQuorum v2.0	No	Yes	No	No
[82]	Fabric v1.2	No	Yes	No	No
[195]	Sawtooth v1.1 with PoET CFT	No	Yes	No	No
[196]	Fabric v1.0	No	Yes	No	No
[191]	Fabric v1.4	No	Yes	No	No
[169]	Fabric v0.6.0 and enterprise Ethereum (Geth) v1.4.18	Yes	Yes	No	No
[151]	Fabric v0.6 and Fabric v1.0	No	Yes	No	No
[194]	Fabric v1.2	No	Yes	No	No
[60]	Sawtooth v1.0.5	No	Yes	No	No
[199]	Fabric v1.0	No	Yes	No	No
[8]	Sawtooth v1.0	No	Yes	No	No
[200]	GoQuorum v2.0.2	No	Yes	No	No
[67]	Fabric v0.6.0-preview, enterprise Ethereum (Geth) v1.4.18, Parity v1.6.0	Yes	Yes	No	No
[118]	Fabric v1.3	No	Yes	No	No
[172]	Fabric, Sawtooth, Burrow, BigchainDB, MongoDB (September 2019)	Yes	Yes	No	No
[22]	Sawtooth v1.1.2, enterprise Ethereum (Geth) v1.8.21, enterprise EOS v1.5.3	Yes	Yes	No	No
[150]	Fabric v1.4.4	No	Yes	No	No
[210]	Fabric v1.4.3	No	Yes	No	No
[204]	Fabric v1.0	No	Yes	No	No
[85]	Fabric v2.0	No	Yes	Yes	No
[188]	Fabric v1.4.4, Sawtooth v1.2, Indy v1.12.0, Parity v2.5.10, GoQuorum v2.3.0, enterprise Ethereum (Geth) v1.9.8	Yes	Yes	No	No
Chapter 4	Fabric v2.2.2, Sawtooth v1.2.3, Besu v21.1, GoQuorum v21.1	Yes	Yes	Yes	Yes

preview, Geth v1.4.18, and Parity v1.6.0 [67]. Other authors analyzed the same frameworks, but with limited industrial relevance because of the employment of a single-node setup which nullifies the impact of consensus and network delays [169]. For both studies, the versions of the frameworks suggest that they are now outdated.

Many authors discuss improvements to the official framework releases. Ref. [196] proposes a BFT algorithm for Fabric v1.0 and analyzes the impact of its employment on the ordering service [196]. Ref. [204] is an in-depth study on Hyperledger Fabric v1.0 and analyzes how configuration parameters affect the performance of the framework. Some of the improvement proposals by the authors were subsequently adopted in Fabric v1.1 [204]. Ref. [82] describes FastFabric, which optimizes Hyperledger Fabric 1.2 to the point of processing almost 20000 transactions

per second, even though some of the proposed measures may be concerning, like storing the state database in volatile memory only [82]. Ref. [118] studies Fabric 1.3 and proposes to optimize the order and validate phases. Tests performed on a four-nodes network with a one-node Kafka orderer show the benefits of the proposals [118]. In industrial contexts, however, performance evaluations on official releases guaranteeing stable and long-term support are preferred.

Some authors analyze a single framework instead of comparing multiple ones. Ref. [16] presents an in-depth performance evaluation of Quorum (GoQuorum client v2.0) with both Raft in a three nodes network and IBFT in a four nodes network using private and public transactions and four workloads [16]. Also other researchers studied Quorum with IBFT, Raft, and Clique consensus with similar objectives [141]. Ref. [143] uses GoQuorum client v2.2.1 with Raft consensus in a network of three nodes deployed once in the cloud, once locally on physical machines, and once on virtual machines [143]. Ref. [210] evaluates Fabric v1.4.3. The authors tested Solo, Kafka, and Raft consenus and the impact of endorsement policies on the overall performance [210]. Ref. [150] introduces a benchmark tool, named HLF-GLDB, simulating the database access patterns of Hyperledger Fabric. HLF-GLDB allowed the authors to discover some bottlenecks of Fabric v1.4.4 [150]. Ref. [85] analyzes Fabric 2.0 in-depth. The study examines network delays, various network sizes and underlying hardware, crashing nodes, private transactions, and multiple workloads [85]. Ref. [195] analyzes the transaction throughput of Sawtooth v1.1 with PoET CFT under different conditions, including network size, network bandwidth, underlying hardware, cloud service, and datacenter location [195]. Such works do not compare different frameworks, but provide meaningful insights into the configuration of a given one.

Some authors evaluate multiple frameworks in terms of performance. Ref. [168] compares Fabric, enterprise Ethereum, Quorum, Corda, and MultiChain in the following dimensions: adoption, performance, community activity, and privacy support. The study, however, lacks an experimental performance evaluation and bases its assessments on the results of other studies [168]. Ref. [145] conducts a performance evaluation of Corda, enterprise Ethereum, Quorum (GoQuorum client), and Fabric by creating networks of various sizes on the Microsoft Azure Platform. Except for Corda, however, the versions of the frameworks are not declared [145]. Ref. [22] compares enterprise EOS (client v1.5.3), enterprise Ethereum (Geth v1.8.21), and Sawtooth v1.1.2. The analysis focuses on support, usability,

documentation, scalability, throughput, and CPU and memory usage [22]. Ref. [172] compares MongoDB and blockchain frameworks (including Sawtooth, Fabric, Burrow, and BigchainDB) for IoT-based applications [172]. The results of these studies are not comparable because they employ different testing methodologies.

Some blockchain benchmark tools are present in the literature. In addition to Blockbanch [67], Bctmark [179] abstracts the underlying blockchain frameworks to improve benchmark portability. Ref. [179] uses Btcmark to test enterprise Ethereum and Hyperledger Fabric. Unfortunately, the tool is still in the experimental stage, and performance metrics and standard workloads are not clearly defined [190]. The Distributed Ledger Performance Scan (DLPS) [188] offers standard workloads that can be submitted to different blockchain networks. The DLPS defines clear metrics for evaluating blockchain systems and matches input and output throughput to maximize the performance of the frameworks. The DLPS supports Hyperledger Sawtooth, Hyperledger Fabric, Hyperledger Indy, Quorum (GoQuorum client), and Ethereum (Parity and Geth clients). Hypeledger Caliper [102] is used by many of the previously discussed studies [145, 16, 141]. Caliper provides predefined workloads and supports multiple frameworks. Ref. [190] summarizes the main benchmark tools for permissioned blockchain frameworks.

All the previously analyzed benchmark tools may generate similar workloads on different frameworks, but they do not set up blockchain networks with similar degrees of security, decentralization, and distribution across different frameworks. Thus, comparing the results obtained with such tools on different frameworks may be misleading: trading security and decentralization for efficiency can enhance the performance of any chosen framework. In Chapter 4 we discuss a cross-framework methodology to reduce the differences among frameworks and enable meaningful performance comparison.

2.3.3 Parallel transaction execution for blockchain performance enhancement

Plenty of studies in the literature address the problem of parallel execution for state machine replication (e.g., [29, 17, 3, 4, 86, 64, 109, 137]). Most of them rely on concurrent execution of non-conflicting commands[185], as such commands are the most frequent in many workloads (e.g., [115, 138, 137]). Nonetheless, some Merke-

lized trees may introduce hidden dependencies among non-conflicting commands (more on this in Chapter 5, thus introducing stricter concurrency requirements.

As previously discussed, high transaction throughput is paramount in industrial scenarios, pushing permissioned blockchain frameworks to be designed with parallel transaction execution in mind. Hyperledger Fabric [9], for example, abandons the standard order-execute processing approach to embrace the more speculative execute-order-validate one: transactions are executed optimistically and, in the validation phase, the conflicting ones are aborted. Transaction re-execution [82] and re-ordering [194] are common techniques to reduce the number of aborted transactions. Block-STM [79], used by Aptos, exploits the write set of aborted transactions to speed-up their re-execution.

Hyperledger Sawtooth [155] implements a scheduler that parallelizes transaction execution based on their dependencies, leading to potential performance improvements. Such an approach is fairly adopted in the literature [7, 213]. However, transactions must declare their read/write sets for the parallel scheduler to speed-up execution. Thus, clients have to pre-execute transactions locally, which may lead to invalid transactions when the blockchain state is concurrently modified by multiple clients [213]. Additionally, application developers must minimize the risk of transactions conflicts by opportunely designing smart contracts.

Transactions can be partitioned in shards for parallel execution also based on static [13] or semantic [186] analysis techniques. Such approaches work well when a prefix tree structure represents the state database, but their generalization to other tree structures may be hindered by the ambiguous state representation problem, which we define in Chapter 5.

PEEP [50] uses locks to protect shared resources and prevent conflicting transactions from creating state inconsistencies, at the cost of introsucing a sequential step for lock acquisition purposes. The authors underline that PEEP must use tree structures that are unaffected by transaction insertion order (i.e., prefix trees).

Serializable Snapshot Isolation (SSI) is another technique used to execute transactions in parallel. In this approach, each transaction accesses a snapshot of the database and commits its writes only if no other transaction has modified the same values. SSI accepts a small chance of unnecessary aborts to avoid more complex conflict-checking techniques. Inserting transaction and conflict information directly

into each block may expedite the validation phase [108]. SSI-based approaches have been proposed by multiple studies [152, 177, 217].

Neuchain [164] exploits transaction parallelism without realying on an explicit ordering phase. Transactions are associated with incremental identifiers to enable deterministic conflict resolution, at the cost of centralization and scalability.

Blockpilot [221] pipelines and parallelizes block execution based on the idea that block proposers and validators have different execution contexts and require varying levels of execution determinism and transaction quantities.

Numerous blockchains keep executing transactions in sequence. Consequently, even though new projects may exploit hardware and software co-design for parallel transaction execution [160], the industry will likely adopt an incremental approach to minimize disruptive changes. In Chapter 5 we describe how we introduced parallelism in the Cosmos SDK whithout introducing breaking changes for existing applications using it. We execute transactions optimistically and in parallel, then we analye their dependencies to re-execute conflicting ones. Even though other researchers use similar strategies [207], our algorithm also addresses the ambiguous state representation problem.

2.3.4 Guidelines for smart contracts

Smart contracts were born to automate and digitalize legal contracts [201]. However, in the blockchain context, smart contracts are code scripts that the peers of blockchain networks execute [30]. Thus, smart contracts refer to two distinct concepts [224, 106]. In particular, Ref. [106] distinguishes between smart contract code, executed by blockchains, and smart legal contracts, which are legal contracts in digital form [106]. The study focuses on the legal aspects of smart contracts, claiming that smart contract code is a fragment of smart legal contracts [106]. Additionally, blockchain-based smart contracts may only provide little benefits from a practical standpoint, as many of the advantages advertised by enthusiasts are not legally meaningful and often misleading [144].

In a context dominated by such hype-driven ambiguities and misconceptions, creating standards for smart contracts is challenging [205]. Additionally, the automation of contracts from a legal perspective poses complex issues and suffers from many limitations, including defining their scope and applicability, addressing

internationalization, and assessing their validity [180]. Redefining existing standards is a first step towards eliminating the incompatibilities and shortcomings between smart contract code and smart legal contracts [53]. Additionally, pinpointing the essential requirements smart contract code must satisfy for legal recognition is an active research topic [68]. Such requirements include coding standards, conflict resolution mechanisms, and universal APIs [142]. In Chapter 6, we tackle smart contracts from a computer science perspective, focusing on the application-level implications of the proposed guidelines.

On a more technical side, transparency, availability, and immutability are some of the main characteristics of blockchain-based smart contracts [76]. Some researchers propose innovative design strategies to overcome some of the current issues, including determinism, transaction order dependency, and exception management [134]. Interestingly, embryonic standardization attempts are currently appearing in the literature [167]. Unfortunately, such works lack generalization because they are tailored to Ethereum [30]. Additionally, Ref. [167] includes some dubious guidelines. For example, terminating contracts based on timers may cause inconsistencies at processing or validation time, as discussed in Chapter 6.

Some standards are designed for specific use cases. For example, the literature provides proposals of standards for altering and undoing smart contracts [139] and for financial smart contracts [25].

On a purely technical side, paradigms and tools for smart contracts, including strategies to reduce gas fees [107], are discussed in multiple studies [95, 224]. Formal descriptions of smart contracts and their features [96], as well as issues and related solutions for programming smart contracts [136, 93], are also available in the literature.

2.4 Conclusion

In this chapter, we contextualized this thesis within the existing literature and covered the core concepts that will be referenced throughout the remainder of this thesis.

In Chapter 3, we will start our discussion on blockchain-based logistic applications by addressing blockchain suitability and introducing an electric vehicle supply chain use case that will be further enriched in the following chapters of this thesis.

Chapter 3

Blockchain adoption

The first challenge companies face in building blockchain-based applications is determining whether blockchain is suitable for a given use case. In this chapter, we examine the essential technological conditions that could hinder the successful adoption of blockchain. Additionally, we introduce a decision-making framework designed to assist managers and decision-makers in determining the suitability of blockchain from a technological perspective. Our aim is to offer decision-makers a practical and easy-to-learn tool that simplifies the technical aspects of the technology, focusing instead on managerial-level issues. The contents of this chapter are based on Ref. [27, 38, 41].

3.1 Introduction

Blockchain is an appealing technological choice in many contexts due to its unique value proposition. Companies are adopting blockchian for various reasons, including fighting censorship, enhancing transparency of business processes, removing intermediaries, and reducing unnecessary costs. However, the promise of quick gains induced by the early success of blockchian-based payment and financial applications, including Bitcoin, has generated a frenetic run for adopting blockchain without much consideration for the compromises and drawbacks imposed by the technology.

To make the matter even worse, blockchain is a complex technology that disrupts existing business models, shifting them towards decentralized paradigms, thus introducing numerous challenges across technical, legal, and economic di-

mensions. Consequently, decision-makers often lack the necessary knowledge to make well-informed choices regarding blockchain adoption, falling pray to common misconceptions in the field [184].

In this context, it is no wonder that blockchain is often selected for inappropriate reasons, even when better alternatives exist [21, 87, 44, 120]. As a result, many blockchain projects have short lifespans, with an average discontinuation rate of approximately one year and a survival rate below 10% [206]. The failure to achieve tangible benefits and the technology's unsuitability for specific business cases are among the primary reasons for these project discontinuations, as indicated by a recent study [174]. An illustrative case is Tradelens, a supply chain management platform supported by IBM and Maersk, which was recently terminated due to its failure to achieve global industry collaboration [74].

Thus, it is imperative to develop a comprehensive framework of standards and tools that simplify managerial decision-making concerning blockchain adoption to enable the creation of successful use cases. This process likely encompasses various factors, including technological, economic, managerial, legal, and human considerations, and will probably take decades to be refined. In this chapter we make a first step in this direction by focusing on the technological aspect.

In this chapter, we address the following aspects.

- We propose a decision-making framework for blockchain adoption that abstracts the complexity of blockchain technology. This framework keeps decision-makers focused on relevant decision drivers, assisting them in understanding when blockchain is applicable, valuable, and preferable to other solutions from a technological standpoint. By leveraging our framework, blockchain adoption decision-making becomes more straightforward, efficient, and less error-prone. Importantly, the framework can be employed without prior blockchain expertise, making it an effective tool for decision-makers who lack the time or skills to delve into the intricacies of blockchain technology.

- We provide a rationale behind each decision driver in our framework, shedding light on concealed caveats of blockchain technology that are often inadequately addressed in existing literature. This discussion offers valuable insights for businesses transitioning to decentralized paradigms. A tool implementing our framework is available on Github [35].

- We examine the application of our framework in a use case inspired by the electric vehicle supply chain of a multinational corporation. This use case represents one of the few supply chains where blockchain adoption can be successful, offering insights into the practical application of our framework.

The remainder of this chapter is organized as follows: Section 3.2 describes the problem tackled in this chapter, Section 3.3 elucidates the blockchain adoption decision-making framework and introduces the Blockchain Adoption Decision Counselor, a tool simplifying the application of our framework. Section 3.4 details the application of our framework to a logistics use case inspired by a multinational corporation's electric vehicle supply chain. Finally, Section 3.5 concludes this chapter.

3.2 Problem statement

Many real-world systems inherently exhibit decentralization. For example, supply chains comprise numerous companies, and each company's actions impact the overall supply chain's performance. Consequently, managing supply chains in a decentralized manner and allowing each company to influence the best strategies for overall improvement is a logical approach.

The advent of blockchain technologies has unlocked opportunities to decentralize data management in systems that previously relied on trusted third parties. However, determining whether blockchain adoption is appropriate poses challenges. Blockchain is a complex technology fraught with hidden trade-offs and challenges [198]. Grasping blockchain's intricacies demands a solid foundation in cryptography (e.g., digital signatures, cryptographic hash functions, zero-knowledge proofs), distributed consensus and state machine replication, non-relational databases, and more. Consequently, decision-makers frequently lack the technical expertise required to make informed choices regarding blockchain adoption. This issue is compounded by numerous misconceptions about blockchain-related topics, even within academic literature [12, 33]. Often, hype becomes the primary driving force behind decisions on blockchain adoption: many blockchian-based projects are discontinued due to the technology's unsuitability for specific business cases and the failure to deliver tangible benefits [174], which underlines the difficulties decision-makers face in deciding

if blockchain is the right technological choice. Hence, a high-level, user-friendly decision-making framework is overdue, one that highlights the merits of blockchain technology and, crucially, the prerequisites for realizing these benefits.

We have developed a framework to assist decision-makers in comprehending when blockchain is a suitable, valuable, and superior solution compared to alternatives. Our framework offers several advantages.

- Streamlined decision process: Our framework simplifies blockchain adoption decisions, saving time.
- Structured methodology: Users follow a structured approach that reduces the risks of adopting blockchain for inappropriate reasons or overlooking critical decision factors.
- Accessibility: The framework is accessible to individuals without prior blockchain knowledge or technical expertise.
- Alternative suggestions: When blockchain is not the optimal solution, our framework provides viable technological alternatives.

We believe that the insights provided in this chapter can contribute to raising awareness about blockchain technology and its genuine value proposition.

3.3 Blockchain adoption decision-making framework

This section describes our decision-making framework for blockchain adoption, which is graphically summarized in Fig. 3.1. The framework helps decision-makers understand if blockchain is a sound technological solution. However, before using the framework, it is important to discuss the main reasons for adopting blockchain.

3.3.1 Blockchain's value proposition

Manipulating data in centralized systems controlled by a trusted third party only requires colluding with such a party. Instead, tampering with a blockchain requires colluding with the majority of its participants. The main value provided by

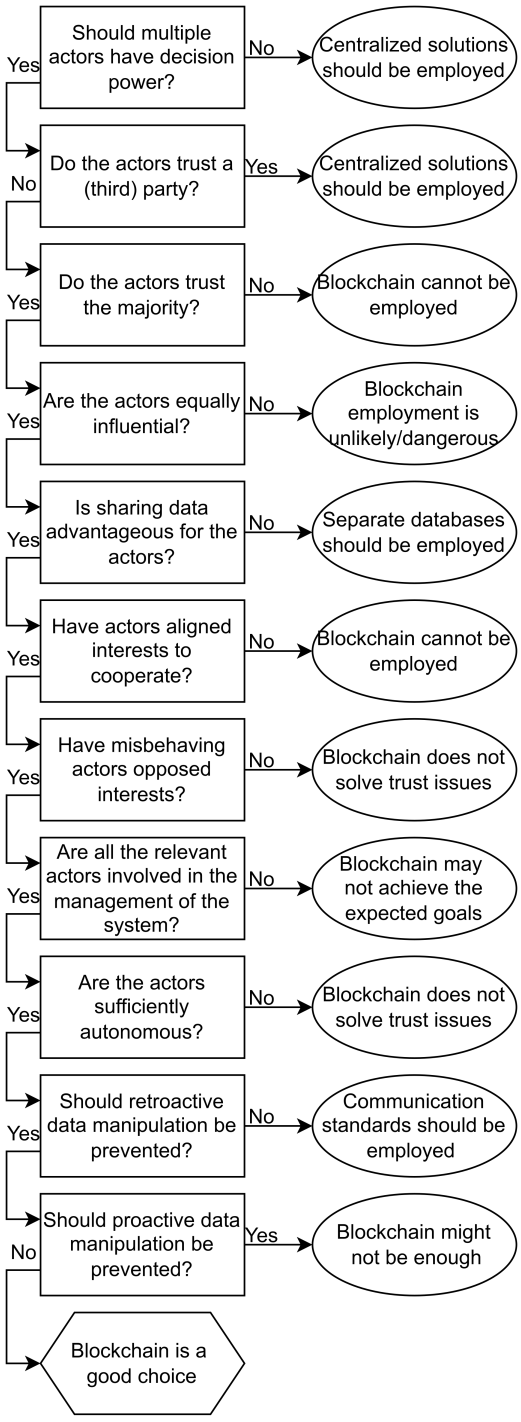


Fig. 3.1 The proposed decision-making framework for blockchain adoption.

blockchain comes from the assumption that the latter scenario is nearly infeasible, making blockchain data trustworthy. Thus, blockchain's value proposition is trust

creation [133]. This, however, can have a few interesting applications. The most immediate one is preventing others from manipulating data, which may be helpful for due diligence practices, preventing censorship, or training machine learning models. Alternatively, blockchain may be used to demonstrate one's inability to manipulate data, which may enhance brand reputation and quality assurance. Finally, sharing a database with multiple other parties enforces a certain degree of standardization, which may simplify and streamline many business processes.

3.3.2 Preliminary remarks

Once blockchain's value proposition aligns with a certain use case, our decision-making framework may be used to assess if the technology is well-suited from a technological standpoint. Nonetheless, a few additional remarks are in order to better understand our framework.

- Blockchain is meaningful when decentralized governance is required. Even though the naming convention. *distributed ledger technology* has gained adoption, decentralization is what matters, not distribution [55].
- Blockchain is inefficient and should be used only when necessary. Blockchain is the only technology allowing for managing a database in a decentralized fashion. However, if the database can be managed by a single entity, other technological solutions are better [48].
- Our analysis focuses on the technological perspective. In some scenarios, blockchain could be preferred to better technologies based on other factors, including marketing and cost-benefit tradeoffs. For example, a company may prefer to pay the transaction fees to deploy a smart contract on a public blockchain instead of sustaining the costs for designing, building, monitoring, and patching a centralized production architecture.

As a consequence of the previous points, fully private blockchains have little to no potential to be used, in our opinion. They can be employed to prevent accidental data modifications, but non-distributed ledgers are more efficient (e.g., ImmuDB [158]). Thus, employing fully private blockchains can be a good marketing strategy but not a good technological one. For example, central bank digital currencies

[23] should not leverage blockchain if they are managed by a single entity (the central bank). Consequently, our framework deals with the suitability of public or consortium blockchains.

3.3.3 Q1: should multiple actors have decision power?

The pivotal consideration in determining blockchain adoption lies in the decentralization of the system. If decision-making authority isn't distributed among multiple entities, it's advisable to favor centralized solutions, such as other distributed databases like Cassandra [122]. This preference stems from the scalability trilemma, indicating that decentralization comes at the expense of either scalability or security [182]. Consequently, centralized solutions exhibit higher scalability and security compared to blockchain.

It's crucial to emphasize that possessing decision-making authority equates to having voting power for validating write attempts in the context of blockchain. Blockchain can be conceptualized as a database subject to alterations through a majority-based voting mechanism. This allows multiple entities to vote and collectively determine the validity of proposed database modifications. Notably, blockchain doesn't extend the same assurances to read attempts, as a single malicious actor could compromise the confidentiality of the database. Thus, while blockchain enhances data integrity and availability, it simultaneously diminishes confidentiality.

Importantly, it's worth noting that validating write attempts doesn't imply having the right to initiate writes. Drawing an analogy to a legal trial, a judge decides the admissibility of evidence but doesn't produce the evidence. This nuanced distinction is a key point of differentiation between our work and existing literature.

3.3.4 Q2: do the actors trust a (third) party?

If an external entity or one of the actors is exceptionally trustworthy, the actors may find it acceptable to delegate their decision-making power to such an entity. In such instances, centralized solutions managed by the trusted party emerge as preferable alternatives to blockchain, echoing the rationale outlined in the preceding section. Conversely, blockchain becomes a viable solution when no single party enjoys the

trust of all actors. This decision driver is a cornerstone in almost all decision-making frameworks found in the existing literature, underscoring its paramount importance.

Notably, a blockchain can itself serve as a trusted third party. For instance, individuals interacting with smart contracts on established blockchain networks (e.g., Ethereum) implicitly place trust in the governance models of these networks.

3.3.5 Q3: do the actors trust the majority?

While blockchain offers notable advantages, it doesn't completely eradicate trust issues. A fundamental prerequisite for utilizing blockchain technology is the trustworthiness of the majority of actors involved. Consequently, blockchain is best suited for scenarios where the likelihood of collusion among actors is low. This precaution is crucial because, in situations where collusion is possible, a malicious majority could exploit the system through tampering or rewriting the database—A phenomenon known as a 51% attack [88]. Unfortunately, existing literature has sometimes overlooked the significance of this critical decision factor.

We strongly advocate for decision-makers to thoroughly assess the potential for 51% attacks before embracing blockchain technology, as these attacks are not uncommon [140, 209]. It's essential to recognize that smaller networks are more susceptible to 51% attacks, requiring fewer actors to collude. Additionally, certain blockchains may be vulnerable to attacks with an even lower percentage of colluding peers due to specific voting protocols. Therefore, especially in consortium blockchains, verifying the presence of a trustworthy (super)majority becomes imperative.

3.3.6 Q4: are the actors equally influential?

For blockchain to emerge as a viable solution, it's crucial that actors maintain a comparable level of decision power. When one actor wields disproportionate influence over others, the likelihood of that influential actor imposing a centralized solution is high. In such instances, adopting blockchain becomes challenging, as the dominant actor has little incentive to share control of the database. Even if a blockchain is implemented, the influential actor could potentially compel others to align with its decisions, raising concerns about the overall trustworthiness of the majority.

In real-world scenarios, achieving a perfect balance of influence is rare and subject to fluctuations over time. To account for this, risk assessment strategies can be employed to consider the possibility of a minority gaining sufficient influence to manipulate others' decisions. Porter's five forces analysis [170] offers a valuable tool for gauging the influence of various actors, with a focus on studying the bargaining power of customers and suppliers. This analysis can provide insights into the dynamics of power among actors and inform decisions regarding the applicability of blockchain technology.

To illustrate, consider Amazon [175], which manages one of the world's largest marketplaces. Sellers benefit from features such as increased visibility and logistical support but are bound by Amazon's non-negotiable policies. In situations where an entity like Amazon possesses significant bargaining power, the adoption of a blockchain solution becomes unlikely, as the powerful entity can dictate the use of its managed database. Conversely, the adoption of blockchain might be plausible in scenarios involving the collaboration of equals, such as the creation of a unified marketplace between Amazon and Alibaba [91], where both e-commerce giants share comparable bargaining power.

3.3.7 Q5: is data sharing advantageous for the actors?

Blockchains serve as shared databases, and participation in a blockchain network implies a willingness to share and receive data. In cases where data is not meant for sharing, centralized databases, where the manager retains complete control, are more suitable. However, blockchain can be harnessed in scenarios demanding unconventional data-sharing methods, and we've identified several such cases: partial sharing, delayed sharing, conditional sharing, and proof sharing.

Partial sharing involves the need to share data with only specific actors. While a separate blockchain with selected receivers is ideal, practical considerations may lead actors to store encrypted data in a unified blockchain, sharing the decryption key exclusively with the intended recipients. This way, encrypted data remains in a tamper-proof database accessible to all actors, but only those possessing the decryption key can recover the original information. Different encryption/decryption keys can be employed to reveal data to distinct subsets of actors.

Delayed sharing allows for data sharing in the future while ensuring non-alteration in the interim. For instance, certain countries unveil classified documents after a predetermined period. Storing encrypted documents in a blockchain and subsequently disclosing the decryption key ensures the authenticity and integrity of the documents at the time of disclosure.

Conditional sharing enables data sharing contingent on a specific event, such as a company sharing confidential data only in the event of litigation. Similar to delayed sharing, blockchain can be utilized to uphold the authenticity and integrity of encrypted data, revealing the decryption key when necessary. Notably, if the anticipated event never transpires, blockchain stores data that remains undisclosed.

Proof sharing involves sharing not the data directly, but a proof computed on the data (e.g., zero-knowledge proofs [92]) or a fingerprint of the data (e.g., the hash of the data [220]). Such approaches aim to guarantee data integrity and minimize information disclosure. Interestingly, blockchain is employed to share data, albeit not in its original form.

3.3.8 Q6: have actors aligned interests to cooperate?

Blockchain systems operate on the principle of majority consensus, a state achievable when actors are motivated to adhere to common rules. For blockchain to be viable, it necessitates a scenario where cooperation is not only advantageous but also willingly embraced by the participating actors [149].

Public blockchains often employ economic incentives to align actors' goals and encourage adherence to predefined rules [159]. In contrast, consortium blockchains commonly rely on indirect incentives, such as business opportunities and cost savings. In domains like logistics, the benefits of data sharing extend to improved demand forecasting, reduced paperwork, and streamlined asset tracking throughout the supply chain. This shared interest creates a foundation for long-term cooperation among actors in the supply chain, fostering an environment where blockchain adoption becomes both feasible and sustainable.

3.3.9 Q7: have misbehaving actors opposed interests?

Even when actors are highly motivated to cooperate, there's a risk that stronger incentives to cheat may emerge, especially in scenarios offering quick and easy gains. To address this concern, it's crucial to ensure that misbehaving actors face conflicting goals, where one actor's gains equate to another's losses. This minimizes the risk of collusion attempts, as actors would need to act against their own interests to corrupt the system.

Consider the example of Bitcoin [192]. Each Bitcoin holder is motivated to create new Bitcoins, increasing their purchasing power. However, this also inflates the existing supply, diminishing the purchasing power of other holders. As a result, Bitcoin holders have conflicting interests when it comes to misbehaving, making collusion attempts unlikely.

We now consider a scenario where a group of friends is betting on the winner of a horse race, assuming that each friend picks a different horse. If the friends opt not to rely on trusted third parties, they might choose to create their own blockchain and utilize a smart contract to collect money in advance and then distribute it to the winner. The verifiability and tamper-proof properties of the blockchain would seemingly assure the correct handling of the bet, making blockchain appear to be a suitable solution. Unfortunately, in this situation, the majority of the friends may end up losing the bet and are likely to collude to reclaim their money instead of forwarding the prize to the winner. Since blockchain decisions, including smart contract behavior, are based on majority agreements, the winner of the bet may not receive the prize. Thus, blockchain should not be employed when cheating attempts favor the majority. However, if the contract were deployed on a public blockchain, the collusion among friends would likely be insufficient, as only a global-scale collusion could invalidate the bet.

3.3.10 Q8: are all the relevant actors involved in the management of the system?

By leveraging blockchain, decisions can be taken through majority voting instead of being delegated to a trusted third party. However, it's essential to recognize that a blockchain system acts as a third party for actors without voting power. Consequently,

blockchain doesn't provide additional trustworthiness guarantees to these actors, and members of the blockchain should not expect entities to acknowledge the trustworthiness of third-party managed blockchain systems.

Taking logistics as an example, consortium blockchains are often utilized to facilitate data exchange among supply chain companies [92]. However, final retail consumers are rarely part of the consortium due to lacking the means, technical knowledge, time, economic incentives, and willingness to be involved. For them, logistic blockchains function as trusted third parties. Consequently, supply chain companies should not join a blockchain system solely to enhance data transparency for final consumers, as consumers have no reason to trust the data stored in a blockchain more than the data provided by their retailer. Thus, blockchain systems should be used to create value for their participants, not external entities.

3.3.11 Q9: are the actors sufficiently autonomous?

The resiliency of blockchain is directly tied to its level of decentralization. To ensure sufficient resilience to errors and tampering attempts, actors within the blockchain should be as autonomous and independent as possible. If too many actors depend on others for tasks such as coding smart contracts, maintaining an updated ledger copy, participating in the voting process, and validating transactions, the blockchain, though seemingly decentralized, becomes substantively centralized. In such a scenario, blockchain provides no advantages over centralized systems but still introduces significant scalability drawbacks. Therefore, it is crucial not to use blockchain if genuine decentralization cannot be guaranteed. Importantly, increasing the number of blockchain nodes managed by each actor does not enhance decentralization, as distribution and decentralization are distinct concepts [55].

Non-autonomous actors pose a threat to decentralization, as they rely on potentially dishonest third parties. Consequently, non-autonomous actors may be exploited to reinforce a dishonest minority. As a result, a small group of autonomous actors offers superior decentralization and security guarantees compared to a larger number of non-autonomous actors.

3.3.12 Q10: should retroactive data manipulation be prevented?

Data stored in a blockchain is considered tamper-resistant as long as the append-only property of the ledger is upheld by the honest majority. This means that once data is inserted, it cannot be manipulated thereafter in a sufficiently decentralized blockchain. Updates are only possible by appending a newer version of the data to the ledger. Importantly, the blockchain retains both versions, enabling actors to track changes. Blockchain proves to be a valuable solution when preventing retroactive data manipulation is crucial. However, if this property is not a priority, simply standardizing data exchange protocols among actors is sufficient for efficient data sharing.

In certain scenarios, companies may opt for decentralized solutions to prevent themselves from engaging in retroactive data manipulation. This strategic choice effectively enhances the transparency and verifiability of the company's operations. For instance, a poker service could leverage blockchain and multi-party computation techniques [223] to ensure the fair extraction of cards, mitigating accusations of favoritism towards any player.

When preventing retroactive data manipulation is not a top priority, each actor can manage its centralized database and utilize standard data-sharing protocols to exchange information with other actors. Interestingly, blockchains are sometimes adopted solely to enforce standardization [156].

3.3.13 Q11: should proactive data manipulation be prevented?

As discussed earlier, blockchain is effective in preventing retroactive data manipulation. However, it has limited capability in preventing proactive data manipulation—Instances where manipulation occurs before the data is stored in the blockchain. This limitation is particularly evident in the case of oracle data, which often conveys information about the physical world and is challenging to verify and validate.

In the realm of logistics, consider the measurement of the temperature of a frozen product at a specific time. Typically, the actor handling the product at that time is responsible for taking this measurement. Other actors, lacking physical possession of the product, must rely on and cannot independently verify the accuracy

of this measurement provided by the handling actor. In such scenarios, blockchain is susceptible to the 'garbage in, garbage out' (GIGO) problem [51].

In contrast, Bitcoin transactions are entirely digital, allowing each peer to independently verify them. Peers can maintain a registry of the balance of each Bitcoin holder, enabling them to determine who has enough coins to spend. Nonetheless, very few use cases can be modeled without relying on oracles, which limits blockchain usefulness when proactive data manipulation must be prevented.

3.3.14 Blockchain Adoption Decision Counselor

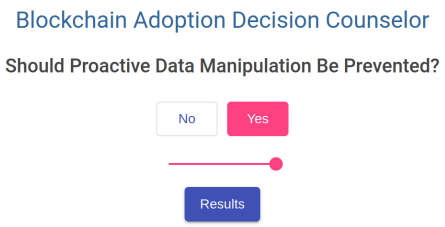


Fig. 3.2 The user interface of the BADC tool

The Blockchain Adoption Decision Counselor (BADC) is a tool that further simplifies the application of our decision-making framework. The tool is implemented by leveraging the Angular framework [154] and is available on Github [35].

BADC exposes a graphical user interface that collects the user's answers and evaluates the blockchain adoption feasibility. The tool moves to the next question as soon as the user answers the current one. Nonetheless, the user may decide to fill in the questions in any order by selecting them through the slider. The results button becomes clickable once all the questions are answered. The results button loads a different view displaying BADC's blockchain suitability evaluation (see Fig. 3.2).

The evaluation view presents BADC's assessment based on suitability, usefulness, and alternatives.

BADC classifies blockchain suitability into three categories: recommended, suboptimal, and discouraged. The recommended category indicates that blockchain is the best technological alternative and that its adoption is encouraged. The suboptimal category indicates that blockchain is usable, but other technologies should be preferred. Nonetheless, other factors (e.g., economic or regulatory ones) could

Table 3.1 Comparison between public and consortium blockchains

	Public	Consortium
Decentralization	Higher	Lower
Development costs	Low	High
Development time	Short	Long
Infrastructure costs	No	Yes
Transaction fees	Yes	No
Flexibility	No	Yes
Computational resources	Shared	Dedicated
Speed	Low	High
Privacy	Whole world	Consortium level
Best suited	Light or intermittent workloads	Heavy workloads, legal compliance

still make blockchain a good compromise. The discourage category indicates that blockchain is not applicable and should be avoided.

BADC also provides an evaluation of the usefulness of blockchain technology. In particular, BADC warns the user if blockchain is not sufficient to provide the expected value proposition, for example, due to the presence of the GIGO problem. In some applications, blockchain may be coupled with additional technologies patching its shortcomings.

In addition to the previous classifications, BADC also suggests potential technological alternatives that may be preferable to the blockchain. Thus, decision-makers may follow the suggestion and change their technological orientation.

3.3.15 Public vs consortium blockchains

Public and consortium blockchains offer different tradeoffs that may be well suited for different use cases. Consortium blockchains may be used when a group of non-trusting entities has the necessity to control who can read/write data to the blockchain (e.g., for compliance with GDPR regulatons). Moreover, they are a

good fit when there are specific requirements for the network configuration, or when a goal throughput must be guaranteed. Usually, consortium blockchains do not require a transaction fee, but each actor must manage one or more nodes (with the related costs and management problems). Such costs are fixed, even if the network is unused. Consortium networks are usually small-sized, which means that transactions can be processed efficiently, but the network is less decentralized and, thus, secure. Public blockchains are a good fit when data can be publicly accessible. Usually, public networks require a per-transaction fee, so they are a good choice when transaction submission is infrequent. Since a single network hosts multiple applications, computational resources are shared and hardly ever wasted. Of course, confidentiality of both public and consortium blockchains can be enhanced through encryption techniques which, however, usually require use case-specific implementations. The main differences between public and consortium blockchains are summarized in Table 3.1,

Let's consider a voting system for political elections. By assuming to put aside some concerns like anonymity, it is possible to describe elections as an event that happens sporadically, but that produces a huge amount of transactions in a short period of time. In such a case, a public blockchain is a good fit:

- The blockchain is used by many different users, not only by voters.
- Voters only need to deploy a smart contract and are not requested to manage a node.
- When there is not an election, voters controlling a node can mine other users' transactions and get a reward.
- In the election period, voters will submit many transactions. Even if none of them owns a node, they can rely on the computational resources of the network in exchange for a transaction fee.
- Even waiting some weeks for the transactions to be processed should not be a problem. Anyway, voters may decide to overpay their transactions in order to speed up the process.
- The election records are public, and the election results can be verified by any interested party.

Let's consider a simple supply chain. Products must be tracked continuously, from a supplier to a consumer, and decisions are taken in real-time based on the state of the products, the availability of carriers and warehouses, etc. In such a case, a consortium blockchain is a good fit:

- Transactions are submitted to the blockchain continuously and at regular rates.
- Transactions must be committed immediately to make real-time decisions.
- Costs are not a concern since the system pays itself in terms of process optimization and litigation reduction (if not, there is no point in using a blockchain).
- Data are not visible to competitors, but are disclosed to the consortium members.

Let's consider the previous example but in the case of seasonal products (e.g., Easter eggs). In such a case, a cost-optimized solution could rely on a public blockchain, since a consortium blockchain would be idle for most of the time. However, based on other restrictions (e.g., GDPR), it could be unadvisable to store data in a public ledger, even in the form of a hash, which is a fingerprint of the data. In fact, since hashes are deterministic and unlikely colliding, they are good identifiers and could be used to track and link data. For example, if different companies store the list of the hashes of the emails of their clients on a public blockchain, a hash present in all the lists would likely imply that the same person is a client of all the companies. Thus, each company would have access to the complete list of relationships entertained by each of its clients.

3.4 Use case: electric vehicle supply chain

This section discusses how we used our decision-making framework to determine the applicability of blockchain technology to a logistic use case based on the electric vehicle supply chain of a multinational company. We conducted such an activity in the context of the Cyber security cOmpeteNCe fOr Research anD InnovAtion (CONCORIDA) project [56].

3.4.1 Use case description

The electric vehicle market is dominated by influential actors. The demand for electric batteries comes in large part from a few multinational companies, which are often battery cell suppliers' biggest clients. Contextually, a limited number of suppliers can fulfill the demand of electric vehicle manufacturers. Similarly, only a few shipping companies can handle the volumes imposed by such a large and distributed supply chain. Hence, the current market conditions impose the creation of long-lasting relationships, as changing partners is not possible.

Nonetheless, the current lack of timely, correct, authentic, and verifiable information may hinder long-term commercial relationships. Companies do not want to pay for the errors of their partners but assessing responsibilities in a fair and verifiable way is difficult when data is scattered across multiple information systems, as reconstructing the sequence of events affecting a given battery or vehicle is not possible. In particular, shocks, high temperatures, and an inappropriate state of charge may cause premature degradation of the batteries.

We worked with a multinational electric vehicle manufacturer to assess the suitability of blockchain in the context of their electric vehicle supply chain. Currently, battery cells are shipped by suppliers to the vehicle manufacturer's battery assembly plant. After being assembled, batteries are delivered to the vehicle assembly plant. A complex process is used to assemble vehicles, which are then shipped to the dealer. External logistic companies handle the transportation of the batteries. Sensors are used to monitor the batteries' temperature, position, vibration level, and charge level. Currently, each company manages its own information system and has limited visibility on the events affecting batteries and vehicles managed by other supply chain companies.

The objective of the use case is to use Industry 4.0 technologies to improve the logistic processes in terms of cost and quality. In particular, the use case presents some requirements.

- Companies want to track batteries and vehicles along the whole supply chain. The tracking system should guarantee the timely, safe, secure, and cost-effective charging, monitoring, and certification of the vehicles (batteries).

- Tracking information should be delivered in real-time to all the supply chain companies, which helps guarantee the safety of the operators and the correctness of the charge level at the moment of delivery.
- Different internal and external operators may recharge the vehicles (batteries) if needed.
- The charging service is provided in predefined areas by mobile or fixed devices.
- Cyberattacks targeting the charging areas put at risk both the vehicles and the power grid. Thus, charging areas should be carefully secured and monitored.
- Data should be shared only with the supply chain companies.
- The system should guarantee data correctness, authenticity, availability, and integrity.
- Access to the companies' systems should be granted to trusted operators and any intrusion should be prevented and mitigated.
- The system should comply with the current regulations (e.g., GDPR).
- A universal source of truth should assign responsibilities to companies in case negative events arise. The responsibility assignment process should be fair, unambiguous, transparent, and auditable.

3.4.2 Value proposition

A blockchain-based tracking system may comply with the majority of the requirements of our use case, as digitally signed transactions would be ordered, timestamped, and recorded across multiple nodes. Thus, a blockchain could guarantee data authenticity, integrity, and availability. Moreover, blockchain's transparency could improve the fairness and verifiability of the responsibility assignment process, and some cyberattacks could be neutralized or mitigated by blockchain's redundancy and resiliency. Thus, a blockchain-based system would allow us to fairly and unequivocally assign responsibilities by linking harmful events, batteries affected, handling actors, and time.

Sharing standardized data through a blockchain system may offer additional benefits to the partners. In particular, the additional data available to the partners may

reduce decision uncertainty, improving the machine learning and optimization techniques applied to demand forecasting and production process scheduling. Moreover, information tracking could be used as a form of health guarantee for refurbished and used vehicles, increasing their market value [203].

Nonetheless, blockchain also introduces many challenges (e.g., technical, economic, and legal). However, a more impelling question should be answered before analyzing the potential benefits and obstacles of designing a blockchain-based solution: can a blockchain be used in the first place? In other words, do the fundamental blockchain assumptions hold in our use case?

We describe the application of our decision-making framework to the previously described use case to answer such a fundamental question.

3.4.3 Decision-making framework application

In this section, we apply our decision-making framework to assess the suitability of blockchain for the examined electric vehicle supply chain.

Q1: should multiple actors have decision power?

A unified information system for the entire supply chain is desirable to prevent data from scattering across multiple information systems. Thus, such a unified system should be democratically managed by the totality of the supply chain companies.

Q2: do the actors trust a (third) party?

Supply chain companies are motivated to avoid responsibilities to reduce economic losses, making them unreliable. Moreover, external companies should not have access to the supply chain data. Thus, we could not identify a party that was trusted by all the supply chain companies.

Q3: do the actors trust the majority?

The current market conditions enforce the creation of long-term relationships. The supply chain companies are likely trustworthy, as losing a client/supplier is likely to

cause severe business drawbacks. We could identify ten different companies in the examined supply chain, each having a strong motivation to guarantee the quality of the final electric vehicle. Thus, the majority is likely trustworthy.

Q4: are the actors equally influential?

Actors have similar bargaining power in the examined use case. A few electric vehicle manufacturers drive the demand for electric batteries, a few battery suppliers can fulfill such demand, and a few logistic companies have the necessary means to handle such volumes. Of course, the supply chain also includes other minor companies which are likely to follow the decisions of the big players as they do not have sufficient influence to oppose them. Thus, the systems results balanced overall, with no single company having enough influence on all the others.

Q5: is data sharing advantageous for the actors?

Data sharing allows for transparently assigning responsibilities and building long-term relationships among companies. Moreover, sharing standardized data may reduce decision uncertainty, improve demand forecasting, and simplify production planning. By detecting failures in the early stages of the supply chain, it could be possible to improve the quality of the final electric vehicles, which could be beneficial to the brand reputation of the supply chain companies.

Q6: have actors aligned interests to cooperate?

Currently, market competition is among supply chains, no longer among organizations [127]. Hence, supply chain companies have a strong motivation to cooperate, as the value perceived by the final consumer is a sum of the value generated by each link in the supply chain. Thus, supply chain companies' performance is tied to the performance of the supply chain as a whole.

Q7: have misbehaving actors opposed interests?

If a company damages the batteries (or the vehicles) and does not take responsibility for it, the economic loss will affect some other member of the supply chain. In the

worst case, the final consumer will purchase a defective vehicle, which would have a negative impact on the reputation of the whole supply chain. Thus, each supply chain company has a strong motivation to prevent selfish behaviors and to request the maximum process quality from the other supply chain members. Thus, misbehaving collusion attempts are unlikely.

Q8: are all the relevant actors involved in the management of the system?

All the supply chain members are meant to participate in the blockchain-based shared information system, except for the final consumers, which lack the proper means and incentives to maintain a blockchain node and increase the level of security and decentralization of the system. Thus, the value of the blockchain system must be measured according to the benefits provided to the battery suppliers, the vehicle manufacturers, and the transportation companies. The benefits to the final consumer must not be considered, even though every supply chain aims at maximizing the value produced for the final consumer.

The goal of the system is to assign responsibilities among the supply chain companies to create long-term cooperation, and all such companies could be included in the management of the blockchain system.

Q9: are the actors sufficiently autonomous?

The supply chain of our electric vehicle supply use case comprises both small and big companies. Big companies have the skills and financial resources to set up and manage their blockchain node, while small companies will likely rely on the services provided by the bigger ones. Nonetheless, the ten big companies of our use case should be sufficient to create a truly decentralized blockchain system.

Q10: should retroactive data manipulation be prevented?

Retroactive data manipulation must be prevented to create a unified and reliable record of the events affecting a given battery (or electric vehicle), which is necessary to fairly, transparently, and verifiably assign responsibilities. Supply chain companies should be prevented from hiding or reassigning responsibilities by tampering with the record.

Q11: should proactive data manipulation be prevented?

Preventing proactive data manipulation is desirable. Supply chain companies should be prevented from storing incorrect or imprecise data in the blockchain system. However, on-field data is obtained by leveraging sensors, which introduces the GIGO problem: companies may manipulate the sensors, displace them, or disrupt the communication between the sensors and the blockchain system to prevent the recording of unfavorable data. If data cannot be reliably collected, responsibilities cannot be correctly assigned.

3.4.4 Blockchain suitability assessment

From our assessment, blockchain can be applied to the examined use case, but it is not enough to guarantee the desired results. We believe that the adoption of a unified information system based on blockchain could determine an improvement over the currently adopted solutions, as it would prevent retroactive data manipulation. The previous assessment can also be obtained through the BADC tool.

Blockchain must be complemented with other technologies to provide the expected benefits. In particular, designing opportune strategies to reduce the likelihood of proactive data manipulation is important. For example, if the cost of manipulating a sensor is higher than the benefit produced by such manipulation, then it is reasonable to assume that the sensor is unlikely to be manipulated. To this extent, Narrowband-IoT technologies and the involvement of telecommunications operators could guarantee a sufficient degree of data reliability. Nonetheless, in this chapter, we do not address the proactive data manipulation problem and keep our focus on blockchain adoption.

Given the positive blockchain suitability assessment, we will discuss the implementation of our solution. In the next chapters of this thesis, we will further develop this use case by tackling additional problems like framework selection, IoT integration, parallel transaction execution, and more.

3.5 Conclusion

Interest in blockchain technology is surging among individuals, countries, and companies. Yet, the path to blockchain adoption is far from straightforward, given its complexity, necessitating a reevaluation of traditional problem-solving approaches from a decentralized standpoint.

This chapter introduced a decision-making framework designed to aid readers in evaluating blockchain adoption from a technological perspective. We explored various decision drivers that illuminate whether blockchain is not only applicable but also valuable and preferable to alternative technologies. Our framework offers a unique resource for decision-makers, encompassing key factors often overlooked in existing literature. Notably, it is accessible to individuals without an in-depth understanding of blockchain intricacies, enabling managers to drive blockchain adoption efficiently within their organizations, sidestepping the need for extensive blockchain expertise.

Through practical application within a logistic use case centered on a multinational electric vehicle supply chain, we discovered that blockchain applications can be advantageous. However, it's vital to recognize that blockchain must be complemented with other technologies, for example to solve the *garbage in, garbage out* problem.

Our framework underscores the crucial relationship between decentralization and blockchain system security. A paramount takeaway is that blockchain should only be considered when a sufficient level of decentralization can be assured.

In Chapter 4 we will take a step forward toward blockchain adoption by analyzing some of the most used frameworks available in the market and discussing a methodology to compare their performances to select the best suited for our electric vehicle supply chain use case.

Chapter 4

Framework selection

After assessing the suitability of blockchian technology, companies are challenged by the problem of choosing the most appropriate blockchain framework to implement their applications. Multiple solutions are currently available on the market, each with its own strengths and weaknesses. Unfortunately, the variety of offered features, the lack of standard benchmarks, and the absence of comparison guidelines make the framework selection choice fairly complex. This chapter analyzes some of the most popular permissioned blockchain frameworks and proposes a methodology to compare their performances to streamline the framework selection process. The contents of this chapter are based on Ref. [34, 27, 40, 41].

4.1 Introduction

At present, the blockchain landscape is swiftly evolving, with blockchain adoption extending beyond the financial sector into various industries, each with unique requirements. Consequently, numerous blockchain frameworks are emerging to address these diverse needs, with new stable releases appearing regularly. However, this constant influx of frameworks and updates poses a challenge for companies trying to navigate technological decisions, as keeping up with emerging technologies and evaluating the impact of new features becomes impractical.

Efficiency is a crucial factor limiting the applications that can leverage blockchain technology effectively. For instance, in industrial IoT applications like logistics,

where processing numerous transactions within a specific time frame is essential [165, 61], a fair evaluation procedure for different blockchain solutions is fundamental.

Despite the growing importance of blockchain, the literature lacks comprehensive comparative analyses of multiple frameworks. Rapid technical advancements render many existing analyses and evaluations outdated. Additionally, numerous articles focus on performance evaluations of a single framework, hindering fair comparisons due to variations in configurations and testing methodologies.

To address these gaps, this chapter presents an updated comparative analysis and a fair performance evaluation of various permissioned blockchain frameworks. The primary contributions include:

- A comprehensive comparative analysis of widely-used blockchain frameworks, including Hyperledger Fabric [9], Hyperledger Sawtooth [155], and ConsenSys Quorum (utilizing both the GoQuorum and Hyperledger Besu clients) [59]. The analysis covers aspects such as governance, maturity, support, latency, privacy, interoperability, flexibility, efficiency, resiliency, and scalability.
- Introduction of a methodology for conducting a fair comparative performance evaluation of different blockchain frameworks. To the best of our knowledge, this methodology is the first to focus on the cross-framework fairness and comparability of tests. In particular, this methodology is innovative, as it allows for minimizing differences among the different frameworks.
- Presentation of one of the most comprehensive cross-framework performance evaluations in the literature. To address gaps in existing research, recent releases of the frameworks were tested. Moreover, to minimize differences among the various frameworks, similar transactions were submitted, and the same underlying hardware was employed. Different blockchain nodes were deployed over the same industrial cloud infrastructure (Amazon AWS).

Thus, our findings provide a thorough overview of the analyzed frameworks, serving as a valuable guide for companies making informed technological choices.

The remainder of this chapter is structured as follows: Section 4.2 introduces some of the most used blockchain frameworks and discusses the problem addressed in this chapter. Section 4.3 presents the comparative analysis and Section 4.4 describes the performance evaluation of the various frameworks. Section 4.5 discusses

blockchain framework selection in our electric vehicle supply chain use case. Finally, Section 4.6 presents our conclusions.

4.2 Background

In this section, we introduce the frameworks analyzed in this chapter and the problem statement.

4.2.1 Blockchain frameworks

Hyperledger Fabric

Hyperledger Fabric [9] is an open-source framework designed to meet prevalent industrial needs, including identity management, role definition, policy establishment, performance optimization, and data confidentiality. Fabric is part of the Hyperledger ecosystem, an open-source community focused on developing stable frameworks for enterprise-grade blockchain deployments [130]. Offering a modular and scalable architecture, Fabric supports smart contracts written in various widely adopted programming languages. For selective data sharing, Fabric allows private transactions (private data collections) and the creation of independent lightweight chains (channels). Fabric currently supports CFT consensus algorithms like Raft, Kafka (deprecated), and Solo (deprecated), with a future plan for BFT consensus [100]. At the time of writing, version 2.3.2 is the latest available.

Fabric distinguishes between two node types: orderers and peers. Peers execute transactions and maintain ledger copies, while orderers construct blocks. Fabric's transaction processing unfolds in three steps:

- execute — Each transaction type adheres to an endorsement policy specifying which peers must execute it. Clients submit transactions exclusively to endorsing peers for scalability at the cost of decentralization. Endorsing peers process the transaction without ledger updates, sending a signed message back to the client for delivery to orderers.
- order — Orderers create blocks by sequencing received endorsed transactions. Blocks are then broadcasted to all channel peers.

- validate — Each peer scrutinizes the correctness of transactions within the received block, updating its ledger copy. Transactions conflicting with preceding ones within the same block are deemed invalid, among other validation checks.

Hyperledger Sawtooth

Hyperledger Sawtooth [155], another integral member of the Hyperledger framework, prioritizes flexibility and separation of concerns, abstracting the application layer from the security layer. This approach facilitates the creation of blockchain systems with replaceable components. Similar to Fabric, Sawtooth supports smart contract composition in multiple programming languages. It incorporates both BFT (PBFT and PoET SGX) and CFT (PoET CFT and Raft) consensus algorithms. Transaction processing in Sawtooth follows the standard order-execute-validate strategy, with transactions grouped into batches. The most recent version, as of the writing of this document, is Sawtooth 1.2.6.

Key modules within the Sawtooth framework include:

- validator component — Responsible for scheduling transactions and ledger management.
- consensus engine — Implements the chosen consensus algorithm.
- REST API component — Simplifies client interaction with the validator component.
- transaction processor (TP) — Implements smart contract logic.

ConsenSys Quorum

ConsenSys Quorum [59], an open-source blockchain protocol rooted in the Ethereum protocol, provides a platform for crafting high-performance permissioned blockchain systems with robust data confidentiality. Compatible with the Ethereum protocol, Quorum facilitates seamless migration of Ethereum smart contracts. Quorum encompasses two projects: one based on the GoQuorum client, originally developed by J.P. Morgan and currently maintained by ConsenSys, and another based on the

Hyperledger Besu client, implemented in Java. GoQuorum supports Raft (CFT), Clique (BFT), and IBFT version 1.0 (BFT), while Besu supports Ethash, Clique, and IBFT (versions 1.0 and 2.0). At the time of this writing, GoQuorum is at version 21.4.2, and Besu is at version 21.1.7.

In instances where both projects share common features, the term "Quorum" is employed generically, with "GoQuorum" and "Besu" specifying the individual implementations. Quorum's transaction processing follows the standard order-execute-validate strategy.

4.2.2 Problem statement

Selecting an appropriate blockchain framework can pose a challenge, given the scarcity of comparative analyses [193]. Our literature review in Sec. 2.3 revealed numerous research gaps.

- Limited availability of comparative analyses.
- Some analyses concentrating on highly specific applications.
- Testing often involves modified framework versions, rendering results less relevant for official, supported releases.
- Divergent testing methods and conditions hinder meaningful comparisons, even qualitatively.
- Outdated analyses that lack current relevance.

Despite these challenges, the significance of evaluating the performance of diverse blockchain frameworks is underscored by the considerable volume of articles devoted to this topic.

In response to these needs, we present a comprehensive methodology for assessing blockchain frameworks in industrial use cases. This methodology serves as the foundation for our performance evaluation of some of the most widely adopted blockchain frameworks in industrial settings.

4.3 Comparative analysis

This section compares multiple blockchain frameworks from a high-level perspective.

4.3.1 Governance

Expounding on the ability to control, coordinate, and direct a blockchain system, governance, as detailed in Sec. 2.2.1 [163], involves decisions shaped by majority voting, with consensus algorithms serving as the voting mechanisms [146]. Analyzing consensus algorithms becomes paramount for comprehending the governance model of a blockchain system, and we have outlined the offered consensus algorithms for each framework in Sec. 4.2.1.

Given its absence of an official BFT consensus implementation, Fabric, despite fully decentralized execute and validate steps, is categorized as a private blockchain. Concerns regarding the reliability of Hyperledger Fabric have been voiced in environments susceptible to ordering service compromise [135]. Additionally, Fabric's state is organized as a flat key-value store rather than a Merkle tree. This structure necessitates querying multiple peers to validate retrieved data, making it impractical for scaling on medium to large networks.

Sawtooth and Quorum, when deployed with a BFT consensus, extend their utility to constructing both public and consortium blockchain systems, thus providing a means for non-trusting parties to resolve their trust issues.

4.3.2 Maturity

Maturity, signifying the production readiness of blockchain frameworks, asserts that Fabric, Sawtooth, and Quorum are all production-ready, as indicated by their documentation and version numbers [100, 58, 105]. Fabric emerges as the most widespread and used technology among the three, evidenced by its plethora of implemented use-cases [65, 129]. Quorum, too, enjoys common industry usage [59, 65]. While Sawtooth sees somewhat less adoption in the industry [65, 129], it has found widespread acceptance in the academic world [166, 43, 28, 111].

4.3.3 Support

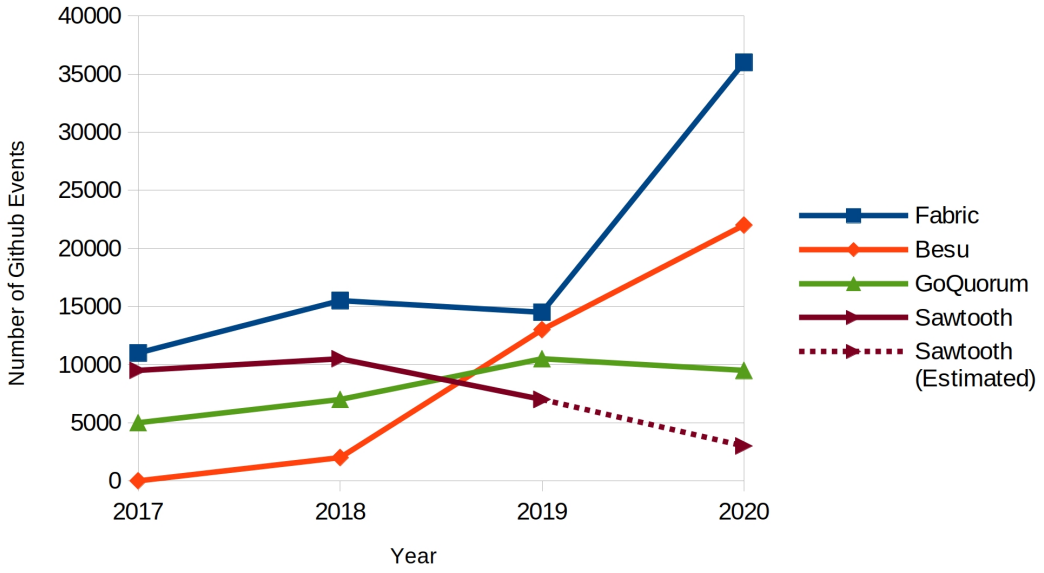


Fig. 4.1 Developer activity (e.g., commits, pull requests, forks, etc.) on GitHub in the years 2017–2020. The image illustrates the most active communities in the blockchain landscape. Sources: [46, 47].

Support, in this context, refers to the extent to which blockchain frameworks facilitate adoption in terms of both technological enhancements and user experience.

Fabric, Quorum, and Sawtooth are active projects, receiving support from both official and unofficial channels.

In our assessment, the Fabric framework is well-documented, though setting up a system from scratch may present non-trivial challenges, warranting potential improvements in official documentation.

In our evaluation, the Sawtooth framework stands out as well-documented and easy to set up, providing comprehensive tutorials for test system setup and detailed configuration options for custom production systems. Sawtooth's abstraction of various blockchain layers makes it an excellent framework for understanding blockchain technology.

In our view, the Quorum framework is partially documented, relying on Ethereum documentation for core concepts. However, it offers many tutorials for test system

setup, focusing on practical aspects. While not as detailed as Fabric or Sawtooth documentation, Quorum's emphasis on practicality is evident.

Regarding community activity, Figure 4.1 depicts GitHub developer analysis from [46] and [47]. Fabric and Besu gain support, GoQuorum remains stable, and Sawtooth experiences a downtrend. As Sawtooth's data for 2020 is absent, we extracted it directly from GitHub, represented by a dashed line.

4.3.4 Latency

Latency, as defined in Sec. 2.2.6, denotes the time taken for transaction processing. Transaction finality can be probabilistic or deterministic based on the consensus algorithm used. Probabilistic finality enhances scalability but introduces higher transaction latency [147]. PBFT [45], IBFT [57], and Raft [157] ensure deterministic finality, while Clique [202], Ethash [214], and PoET (CFT and SGX) [99] offer probabilistic finality.

4.3.5 Privacy

Privacy, in this context, pertains to the ability to share data with a subset of blockchain system participants. Various strategies are applied [155]:

- Share the hash of the data with all peers, and the actual data only with those of interest. This underlies Fabric's private data collections [211] and Quorum's Orion and Tessera modules [2].
- Create a separate system—usually costly and potentially insecure. Fabric addresses this through channels, separate blockchains with their own ledger, sharing common components and reducing hardware requirements compared to separate systems [9].
- Store ciphered data, which is feasible but requires client-managed encryption. Sawtooth exclusively adopts this strategy [155].

4.3.6 Interoperability

Interoperability, referring to the atomic transfer of data across multiple blockchains [121], remains unaddressed by the analyzed frameworks. Cross-chain communication, akin to interoperability but without atomicity, is also absent [219].

Partial interoperability may be achieved using cross-chain communication protocols or relying on game theory and time synchronization or third-party trust assumptions. Protocols like notary schemes or hash-locking [19] have drawbacks, including limited use cases, inefficiency, third-party trust, and atomicity violation. Moreover, they do not facilitate the transfer of asset history across different blockchains, hindering transparency and verifiability.

Full blockchain interoperability is unattainable without ledger merging [121], and cross-chain communication necessitates trusted third parties [219].

4.3.7 Flexibility

Flexibility, denoting the capacity to replace components or add features, varies among the frameworks.

Sawtooth stands out as the most flexible, allowing dynamic component replacement and on- and off-chain settings configuration [155]. Quorum and Fabric also offer flexibility, supporting configurable parameters and plugins [58, 101, 9].

4.3.8 Efficiency

Efficiency measures the amount of information a blockchain framework can process per unit of time. Key insights from the in-depth analysis in Section 4.4 reveal that the choice of a smart contract programming language significantly impacts overall performance. Notably, Fabric and GoQuorum demonstrate exceptional performance across various tests. While Besu performs well with light transactions, it experiences notable performance decay with heavier tasks. Sawtooth's efficiency shows improvement with larger transaction batches, though its parallel scheduler potential remains underutilized due to the limited number of vCPUs utilized in our tests [65, 129, 155].

4.3.9 Resiliency

Resiliency, reliant on consensus algorithms, distinguishes between CFT and BFT. Fabric offers CFT algorithms, while Sawtooth and Quorum provide both CFT and BFT options. Notably, quantum-safe elliptic-curve cryptography is currently absent in all the frameworks [18].

4.3.10 Scalability

Scalability denotes the ability to expand the size of a blockchain network while mitigating adverse effects on other system properties, such as efficiency. As outlined in the scalability trilemma, enhancing scalability in a blockchain system often involves a tradeoff with decentralization or security [165]. A notable example of this tradeoff is the preference for CFT over BFT algorithms, a choice embraced by Fabric, Quorum, and Sawtooth.

Fabric employs endorsement policies as an alternative method to enhance scalability. By allowing different nodes to execute distinct sets of transactions in parallel and leveraging channels, Fabric essentially adopts an approach akin to creating separate blockchain systems [9].

Quorum's IBFT consensus algorithm introduces dynamic changes to the set of nodes participating in the consensus protocol. This feature enables the maintenance of a small, efficient set of consensus nodes, while affording all peers the opportunity to be part of this set for a limited duration [57].

In the case of Sawtooth, the set of consensus nodes in its PBFT is an on-chain setting that can be dynamically updated. This mechanism results in a behavior similar to that of Quorum's IBFT, offering adaptability in response to evolving network dynamics

4.4 Performance analysis

As articulated in Sections 4.1 and 2.3, the absence of a standardized methodology for comparing the performance of multiple blockchain frameworks has complicated the

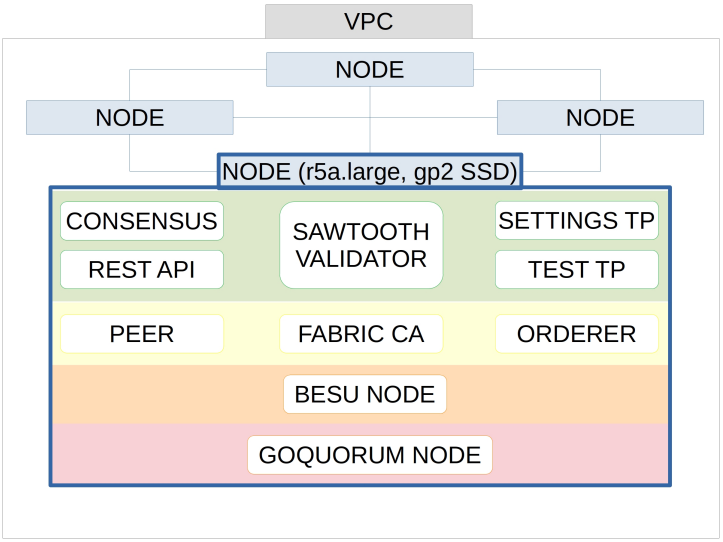


Fig. 4.2 The testing environment architecture involved creating a network of four virtual machines on AWS, with each virtual machine hosting multiple components. Components of the same framework were identified by a consistent color. The Sawtooth node comprised a validator, a consensus engine, a REST API, and two transaction processors—one for managing on-chain settings and another for processing test transactions. The Fabric node included a peer, an orderer, and a certificate authority. In contrast, both the GoQuorum and Besu nodes each consisted of a single component

decision-making process when choosing an appropriate blockchain. This challenge is exacerbated by the limited interoperability among different blockchain solutions.

This section elucidates the testing environment and outlines the conducted tests on various frameworks. Notably, the tests were executed using identical virtual machines, while the smart contract had to be implemented individually in each framework. Furthermore, the configurations of the diverse frameworks were not fine-tuned. This decision was made due to variations in configuration settings among different frameworks, each of which can significantly alter the system's behavior.

In terms of the frameworks incorporated in the tests, we selected some of the most widely-used blockchain frameworks: Hyperledger Fabric, Hyperledger Sawtooth, and ConsenSys Quorum (featuring both the GoQuorum and Hyperledger Besu clients).

Table 4.1 Test environment for each blockchain framework, highlighting the main differences among the various frameworks.

	Fabric	Sawtooth	Besu	GoQuorum
Version	2.2.2 (Jan, 2021)	1.2.3 (Oct, 2019)	21.1 (Feb, 2021)	21.1 (Feb, 2021)
Components per instance	1 peer, 1 orderer, 1 Fabric Certificate Authority	1 validator, 1 consensus engine, 1 REST API, 1 settings transaction processor, 1 test contract transaction processor	1 Besu node	1 GoQuorum node
State Database	LevelDB	LMDB	RocksDB	LevelDB
Consensus	Raft	Raft, PBFT	IBFT 2.0	Raft, IBFT 1.0
Smart Contract	Go, Java	Go	Solidity	Solidity
Batch Size	-	1 transaction	-	-
Number of channels	1	-	-	-
Endorsement Policy	All peers must endorse each transaction	-	-	-

4.4.1 Testing environment

To perform the tests, we constructed a network consisting of four AWS instances. The instances belonged to the same availability zone and to the same virtual private cloud (VPC). Each instance was a r5a.large virtual machine, with 2 vCPUs, 16 GB of RAM, and 50 GB SSD. The testing environment infrastructure is shown in Figure 4.2.

The following settings describe the test environment used for the performance evaluation.

- Network topology: complete graph, with instances hosted in the same availability zone.
- Number of instances: 4.
- Instance type: AWS r5a.large.
- DISK (single instance): 50 GB gp2 SSD (EBS volume).
- RAM (single instance): 16 GB.

- CPU (single instance): AMD EPYC 7000, 2.5 GHz, and 2 vCPUs.
- OS: Ubuntu 20.04.2 LTS.
- Node: v10.24.
- Go: go1.13.
- Docker: 20.10.3, build 48d30b5.
- Docker-compose: version 1.28.4, build cabd5cfb.
- Solidity: 0.8.0+commit.c7dfd78e.Emscripten.clang.
- Java: openjdk v1.8.0_292.

4.4.2 Methodology

Table 4.2 Transaction parameter configurations for various test types. The concurrency test involved accessing a varying number of different addresses, ranging from one to the total number of submitted transactions. In the size test, transaction payload sizes ranged from 0.1 to 50 kB. The iteration test encompassed a variation in the number of read and write operations, spanning from 1 to 1000.

Test	No. addresses	Payload size (kB)	No. iterations
Size	max	0.1; 1; 10; 20; 50	1
Concurrency	1; 100; max	0.1	1
Iteration	max	0.1	1, 10, 100, 1000

As discussed in Section 4.3.10, blockchain frameworks offer the option to trade off decentralization and security in favor of efficiency and scalability. Establishing the performance superiority of one framework over another is straightforward when one is configured for scalability, and the other prioritizes security and decentralization. Consequently, comparing the performance of various frameworks becomes meaningless unless comparable conditions are ensured for all frameworks. While guidelines for evaluating the performance of a single framework exist [103], and some multi-framework benchmark tools have been developed [67, 188], these

resources lack a methodology for creating equivalent testing environments for comparing the performance of different blockchain frameworks. To bridge this gap, we introduce a novel methodology addressing the following concerns.

- Node functional requirements: Frameworks consist of multiple modules that need allocation to hardware resources. However, some frameworks exhibit greater modularity than others. Hence, defining a blockchain node based on its functional requirements becomes crucial. This enables the creation of classes of equivalent modules across diverse frameworks, facilitating consistent module assignment to hardware resources. To our knowledge, no prior study has tackled this issue.
- Distribution requirements: it is imperative to employ identical network topologies and geographic node distributions across various frameworks. It is noteworthy that enforcing uniform geographic distribution becomes feasible only after establishing a cross-framework definition of a blockchain node.
- Resiliency requirements: similar security, decentralization, and replication levels must be mandated across different frameworks, especially concerning the execution of smart contracts and participation in the consensus protocol.
- Number of ledgers: determining the fixed number of separate ledgers managed by each blockchain system and specifying the workload each ledger undergoes is essential. Deploying multiple ledgers offers a straightforward approach to enhancing the throughput of a blockchain system.
- Standardized workloads: designing a suite of standardized tests is necessary to assess the upper bound of a generic production system's performance. Hyperledger Caliper and the DLPS also utilize standardized workloads for benchmarking purposes.

Regarding the functional requirements of nodes, Fabric distinguishes among ordering, endorsing, and validating nodes. Similarly, Sawtooth separates the layers for conensus, ledger management, and smart contract processing. In contrast, a single Quorum node performs all three operations. Consequently, we present an abstract definition of a blockchain node based on its functional requirements, enabling the consistent assignment of framework modules to hardware resources across

diverse frameworks. An abstract blockchain node signifies a non-trusting entity in a blockchain network and should independently perform all pertinent operations. We define an abstract blockchain node as a set of components executing the following tasks:

- security management, covering cryptographic operations and privacy management.
- transaction management and Smart Contract Execution, encompassing ordering, scheduling, and processing transactions;
- peering and networking management (i.e., networking with other peers);
- consensus management (e.g., mining, fork resolution);
- database management, inclusive of ledger and state database updates;

For each framework, we formed networks consisting of four nodes, with each node assigned to a distinct virtual machine. This ensured a one-to-one correspondence between virtual machines and nodes. Subsequently, we allocated modules to virtual machines, aligning with the functional requirements defined for an abstract blockchain node, as depicted in Figure 4.2 and detailed in Table 4.1. Notably, our tests exclude private transactions, making a single Besu or GoQuorum node sufficient to meet the abstract blockchain node criteria. Sawtooth, as discussed in Section 4.2.1, incorporates various modules, with a Sawtooth validator consistently connected to the transaction processor managing on-chain settings. In our deployment, we colocated the consensus engine and the transaction processor for test transactions on the same virtual machine.

Additionally, we established four Fabric organizations, representing non-trusting parties. For each organization, we deployed one peer and one orderer on a single virtual machine. This configuration aligns with the abstract node definition, where a single node is responsible for both transaction processing and consensus management.

Concerning geographic distribution, all nodes for each framework were deployed within the same Virtual Private Cloud (VPC) and maintained full connectivity.

Concerning resiliency requirements, all nodes were mandated to actively participate in both consensus and transaction execution. In Fabric, this translated to the

endorsement of transactions by all four nodes, ensuring each transaction underwent execution precisely four times—Once per node. Where feasible, equivalent consensus algorithms were employed across frameworks, with Raft implemented by all of them. Additionally, PBFT and IBFT exhibited similar behavior when consensus nodes were not subject to dynamic replacement.

Concerning the number of ledgers, each blockchain system was assigned a single ledger, leading to the use of a single channel in Fabric.

For workload simulation, various scenarios were considered:

- transactions writing varying amounts of data to the ledger: this scenario mimics situations where different IoT devices contribute data to the ledger.
- presence of parallelizable and sequential transactions: sequential transactions, typical in step-by-step processes, coexisted with parallel transactions, common in scenarios involving multiple independent processes (e.g., sensors monitoring diverse assets simultaneously).
- transactions updating varying numbers of objects: for example, a single sensor monitoring a cargo might need to simultaneously update data related to one specific good or all shipped goods.

For the performance evaluation, a singular transaction type was defined, encompassing the following operations:

- loading a data structure from the ledger, comprising a counter and a string.
- incrementing the counter and replacing the string with its payload.
- storing the modified data structure back to the ledger at its original address.
- iteratively repeating these steps for a predetermined number of iterations.

Consequently, each transaction was characterized by the following parameters.

- Blockchain address, denoting the location where the data structure is stored. Transactions targeting the same address generated a sequential workload, while transactions targeting different addresses generated a parallelizable workload.

- Payload size, specifying the amount of data to be copied in the data structure modified by the transaction.
- Number of iterations, determining how many times the transaction continues loading and storing data.

Three types of tests were conducted on the frameworks, each focusing on one of the aforementioned parameters.

- Size test: transactions read from and wrote to the ledger with varying amounts of data. The minimum payload size during the tests was set to 0.1 kB, considering that a single hash is usually no shorter than this value.
- Iteration test: each transaction performed a varying number of load and store operations, simulating transactions updating the state of one or more assets.
- Concurrency Test: transactions read from and wrote to a varying number of different addresses. This allowed observation of the frameworks' behavior when transactions were sequential (i.e., reading from and writing to the same address) or parallelizable (i.e., reading from and writing to completely different addresses).

Table 4.2 summarizes the configuration used in each test. Each test was repeated ten times with each set of parameters. Transactions were submitted to one of the four nodes at a rate of 500 transactions per second (tps). This input rate was chosen to surpass the maximum throughput achieved by the frameworks, thereby highlighting their distinct behaviors under the same workload. Performance was measured by a client external to the blockchain system under test. Time measurement by the client extended from transaction submission to transaction consolidation, occurring after a single block confirmation, as deterministic consensus algorithms were employed. It is crucial to note that our objective was to measure the frameworks' performance under similar conditions, and the obtained results do not represent the maximum throughput of the frameworks. Determining maximum throughput typically involves addressing non-polynomial, complex maximization problems that are rarely solved exactly [62].

4.4.3 Environmental similarities and limitations

The frameworks were tested on the same hardware. Moreover, the configurations of the frameworks were not tuned. Depending on the programming languages supported by each framework, similar smart contracts were written in Go, Java, and Solidity. However, some differences existed, due to the unique APIs offered by each framework. The main differences between the frameworks are reported in Table 4.1. For each framework, the table describes the version, the components instantiated on each virtual machine, the consensus protocol, the programming language used to implement the smart contracts, the default state database, the batch size (for Sawtooth), and the endorsement policy and the number of channels (for Fabric).

4.4.4 Results

This section presents the results obtained from the performance evaluation.

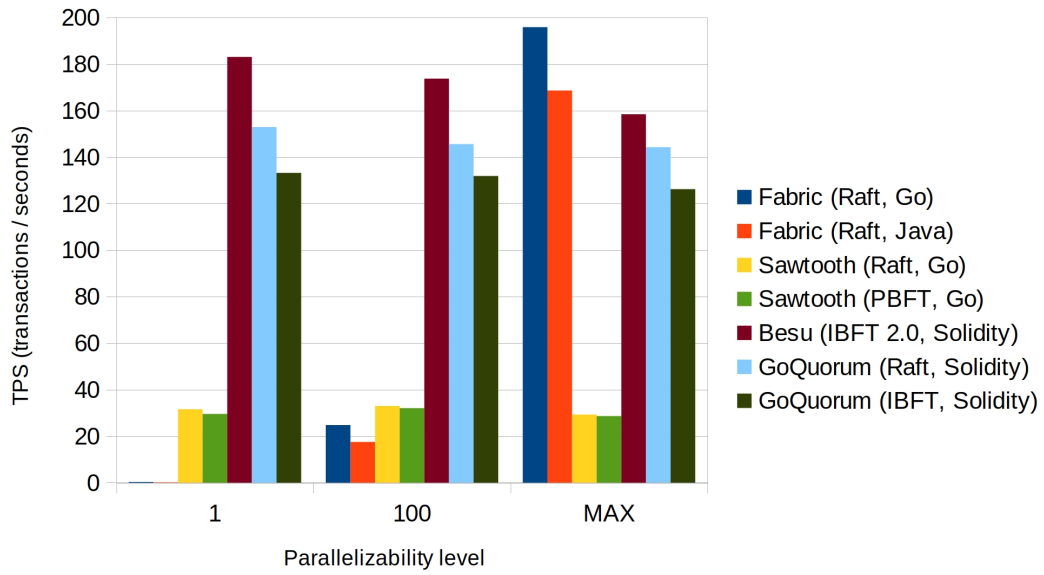


Fig. 4.3 Transactions per second (TPS) measured for varying levels of transaction parallelizability: sequential transactions (parallelizability = 1), transactions with partial parallelizability (allowing up to 100 parallel transactions), and fully independent transactions (max parallelizability).

The results of the concurrency test are shown in Figure 4.3. Fabric did not perform well for sequential transactions: many of the transactions failed the validation step, as

explained in Section 4.2. However, in the vast majority of use-cases, transactions are parallelizable, and both Fabric and Quorum performed well. Sawtooth's performance was affected by the choice of small batches and attained a TPS value half that achieved by Fabric. Moreover, Sawtooth's parallel scheduler did not provide any benefit. This was likely due to the choice of AWS instances with only two vCPUs. For Fabric, the choice of smart contract programming language was important, as those written in Java did not perform the same as the ones written in Go. CFT consensus algorithms boosted performance in all the frameworks, but on small networks, such as the one used for the tests, the performance gain did not justify the sacrifice of decentralization. However, by increasing the number of nodes, the performance advantages of using CFT algorithms on fully connected networks should become considerable, as they have lower message complexity. As the number of exchanged messages is relevant and not the total number of nodes, performances are unlikely to decay on big networks if each node is connected to a limited number of peers. This strategy is adopted by probabilistic consensus algorithms and impacts latency and finality instead of efficiency.

The results of the concurrency test are showcased in Figure 4.3. Fabric exhibited suboptimal performance for sequential transactions, with a notable number failing the validation step (refer to Section 4.2 for detailed discussion). Nevertheless, in scenarios where transactions could be parallelized, both Fabric and Quorum demonstrated satisfactory performance. On the other hand, Sawtooth's performance, influenced by the use of small batches, resulted in a TPS value half that achieved by Fabric. Furthermore, the parallel scheduler in Sawtooth did not yield discernible benefits, possibly due to the choice of AWS instances with only two vCPUs.

In the case of Fabric, the choice of the smart contract programming language played a crucial role, with Java-written contracts exhibiting different performance characteristics than those written in Go. Across all frameworks, CFT consensus algorithms consistently outperformed BFT ones. However, on small networks like the one employed for the tests, the performance gain did not justify sacrificing decentralization. Nevertheless, scaling up the number of nodes could make the performance advantages of CFT algorithms significant, as their lower message complexity is likely to play a crucial role in large and fully connected networks.

It's worth noting that performance is unlikely to deteriorate on larger networks if each node is connected to a limited number of peers, as the number of exchanged

messages, rather than the total number of nodes, is the primary performance limiting factor. This strategic approach aligns with probabilistic consensus algorithms, impacting latency and finality rather than efficiency.

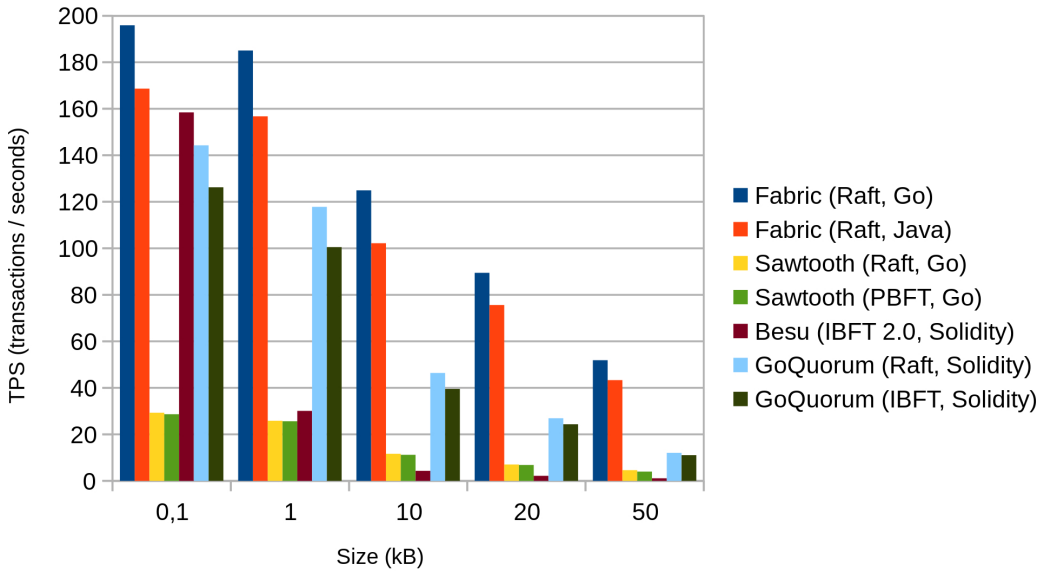


Fig. 4.4 TPS with varying payload sizes, ranging from 0.1 to 50 kB.

Figure 4.4 illustrates the outcomes of the size test, validating the trends identified in the concurrency test. Notably, Besu exhibited a swift degradation in performance with larger transactions. In general, the increase in transaction payload size corresponded to a decrease in TPS, as the volume of data stored per second escalated.

Figure 4.5 showcases the results of the iteration test, affirming the observed performance deterioration of Besu with longer-lasting transactions. Overall, as the number of load and store operations per transaction increased, the quantity of read and write operations per second also rose, albeit with a decrease in Transactions per Second (TPS). Additionally, none of the frameworks appeared optimized for multiple read and write operations on the same address within a single transaction. In such scenarios, it is advisable to perform only the first read and the last write operations, a consideration worth noting in smart contract development.

Our performance evaluation results deviate from those in other studies, a common occurrence due to variations in tools, configurations, and testing methodologies. To mitigate variability, we compare our results to studies employing official framework

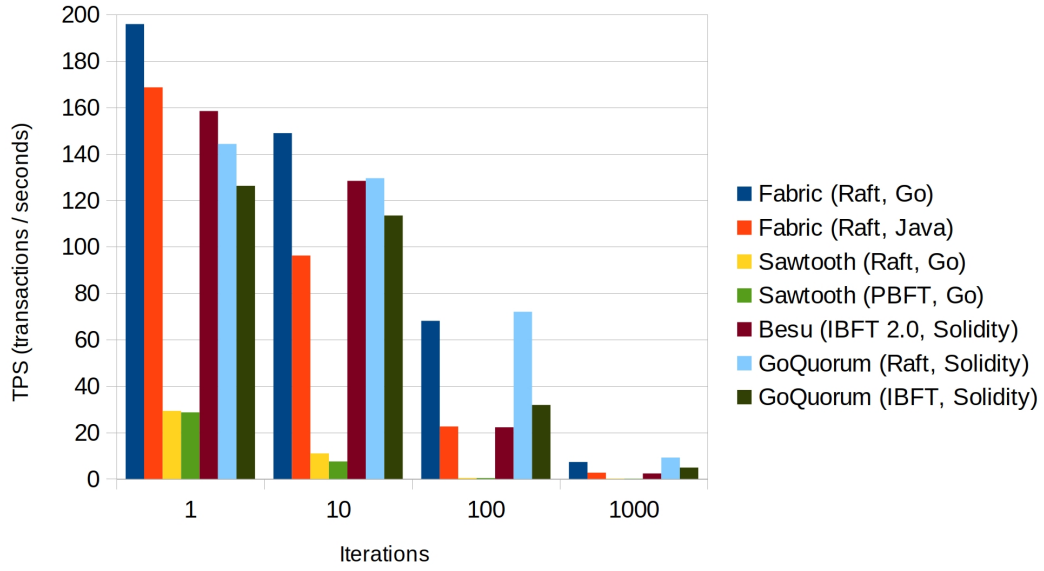


Fig. 4.5 TPS with varying read/write amounts. A pair of read/write operations (iteration = 1) mimics a transaction updating a single asset, whereas multiple read/write operations mimic a transaction updating multiple assets.

versions and discard studies using outdated versions due to potential technological disparities.

Ref. [188] reported superior transaction throughput results across all frameworks and we attribute the variance to more powerful hardware usage. Additionally, even minor differences in framework configurations may significantly impact system performance; for instance, we observed GoQuorum's performance doubling when logging is disabled.

Ref. [85] focused on Fabric exclusively, adopting an adaptive-based testing methodology that aligns input and output transaction rates. However, such a strategy poses challenges in cross-chain comparisons, as diverse frameworks contend with distinct input transaction rates. We opted for a uniform workload (in terms of transaction rate) across different frameworks. Furthermore, the study employed eight peers instead of our four, potentially doubling overall throughput. While numerical values differ, some similarities in overall framework behavior, especially performance decay with increasing payload size, persist.

Ref. [141] did not specify the version of Quorum used, thus the release of Quorum may be outdated even though the study is recent. Differences in transaction

complexity with respect to this work are likely present and due to Caliper usage. The study employed a single, high-performance virtual machine, differing significantly from our distributed four-node setup. Consequently, while certain results may align (e.g., 4-nodes Raft and 4-nodes IBFT), methodological distinctions preclude generalizations.

Table 4.3 Results of the performance evaluation. Each row represents one of the tests performed. For each test, the configuration used and the results obtained are reported.

No. addresses	Payload size (kB)	No. iterations	Fabric (Raft, Go)	Fabric (Raft, Java)	Sawtooth (Raft, Go)	Sawtooth (PBFT, Go)	Besu (IBFT 2.0, Solidity)	GoQuorum (Raft, Solidity)	GoQuorum (IBFT, Solidity)
1	0.1	1	$(3.0 \pm 0.2) \cdot 10^{-1}$	$(1.7 \pm 0.3) \cdot 10^{-1}$	$(3.1 \pm 0.1) \cdot 10$	$(2.9 \pm 0.1) \cdot 10$	$(1.8 \pm 0.3) \cdot 10^2$	$(1.2 \pm 0.03) \cdot 10^2$	$(1.33 \pm 0.07) \cdot 10^2$
100	0.1	1	$(2.4 \pm 0.2) \cdot 10$	$(1.7 \pm 0.3) \cdot 10$	$(3.2 \pm 0.1) \cdot 10$	$(3.2 \pm 0.5) \cdot 10$	$(1.7 \pm 0.4) \cdot 10^2$	$(1.46 \pm 0.05) \cdot 10^2$	$(1.32 \pm 0.05) \cdot 10^2$
max	0.1	1	$(1.95 \pm 0.02) \cdot 10^2$	$(1.68 \pm 0.02) \cdot 10^2$	$(2.9 \pm 0.2) \cdot 10$	$(2.8 \pm 0.4) \cdot 10$	$(1.6 \pm 0.5) \cdot 10^2$	$(1.44 \pm 0.04) \cdot 10^2$	$(1.26 \pm 0.08) \cdot 10^2$
max	1	1	$(1.85 \pm 0.02) \cdot 10^2$	$(1.56 \pm 0.02) \cdot 10^2$	$(2.5 \pm 0.1) \cdot 10$	$(2.6 \pm 0.2) \cdot 10$	$(3.0 \pm 0.6) \cdot 10$	$(1.18 \pm 0.06) \cdot 10^2$	$(1.01 \pm 0.07) \cdot 10^2$
max	10	1	$(1.24 \pm 0.03) \cdot 10^2$	$(1.02 \pm 0.03) \cdot 10^2$	$(1.1 \pm 0.8) \cdot 10$	$(1.1 \pm 0.4) \cdot 10$	(4 ± 1)	$(5 \pm 2) \cdot 10$	$(4 \pm 1) \cdot 10$
max	20	1	$(8.9 \pm 0.3) \cdot 10$	$(7.5 \pm 0.2) \cdot 10$	(7 ± 5)	(7 ± 2)	(2.2 ± 0.5)	$(2.6 \pm 0.8) \cdot 10$	$(2 \pm 1) \cdot 10$
max	50	1	$(5.1 \pm 0.5) \cdot 10$	$(4.3 \pm 0.1) \cdot 10$	(4.6 ± 0.5)	(4 ± 1)	(1.1 ± 0.5)	$(1.2 \pm 0.4) \cdot 10$	$(1.1 \pm 0.5) \cdot 10$
max	0.1	10	$(1.48 \pm 0.05) \cdot 10^2$	$(9.6 \pm 0.2) \cdot 10$	$(1.0 \pm 0.1) \cdot 10$	(7 ± 1)	$(1.3 \pm 0.2) \cdot 10^2$	$(1.30 \pm 0.03) \cdot 10^2$	$(1.13 \pm 0.05) \cdot 10^2$
max	0.1	100	$(6.8 \pm 0.6) \cdot 10$	$(2.25 \pm 0.93) \cdot 10$	$(4.07 \pm 0.02) \cdot 10^{-1}$	$(3.7 \pm 0.1) \cdot 10^{-1}$	$(2.2 \pm 0.3) \cdot 10$	$(7.2 \pm 0.3) \cdot 10$	$(3.18 \pm 0.02) \cdot 10$
max	0.1	1000	(7.2 ± 0.1)	(2.63 ± 0.05)	$(1.26 \pm 0.02) \cdot 10^{-1}$	$(1.12 \pm 0.03) \cdot 10^{-1}$	(2.3 ± 0.1)	(9 ± 1)	(4.9 ± 0.5)

The results obtained are additionally detailed in Table 4.3. Each row in the table corresponds to one of the conducted tests, providing information on the configuration employed and the resulting outcomes.

4.5 Use case: electic vehicle supply chain

Our electric vehicle supply chain use case involves a set of requirements related to operational efficiency, privacy, and regulatory compliance. Based on the discussions in this chapter, permissioned blockchain platforms may be a suitable option as they address the typical needs of industrial applications, offering efficiency, adaptability, and the ability to restrict network access to designated members. Now, we need to choose the most suitable framework based on our analysis.

We may eliminate Hyperledger Fabric due to concerns about data verifiability resulting from using a flat key-value store to represent state and resilience against potential malicious actors due to the absence of BFT algorithms. Security and decentralization are key factors in our use case.

Considering framework performance, we may discard Sawtooth; while its performance is likely sufficient, opting for more efficient frameworks might be wiser.

Lastly, we may rule out GoQuorum in favor of Besu, which is more supported and has a more active community on Github. Consequently, it may receive patches and updates for longer periods. Additionally, many tools developed within the Ethereum ecosystem can be used with Quorum too, enabling faster application development, testing, and deployment.

4.6 Conclusion

The interest blockchain is gaining is prompting a response from the market, with many frameworks for decentralized application development appearing and receiving multiple updates and enhancements. Selecting the most appropriate one becomes difficult due to the lack of fair comparison methodologies and tools. In this chapter, we analyzed Hyperledger Fabric, Hyperledger Sawtooth, and ConsenSys Quorum (with both the GoQuorum and the Hyperledger Besu clients), which are some of the most used frameworks currently available, from multiple perspectives. Additionally, we presented a methodology for fairly comparing the performance of blockchain frameworks. Hopefully, this methodology will help companies in performing unbiased evaluations of existing frameworks.

In Chapter 5, we will extend this discussion by analyzing a possible way to enhance the performance of blockchain frameworks, given that this is presently one of the main barriers to blockchain adoption.

Chapter 5

Boosting performance

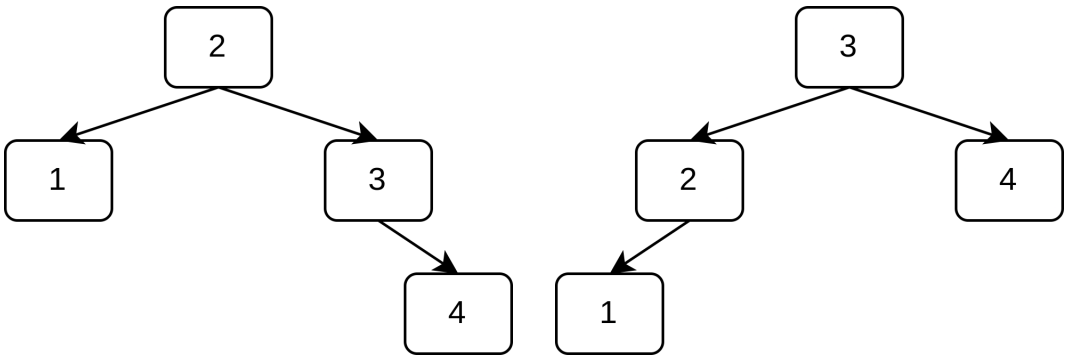
The previous chapter focused on comparing blockchain frameworks, particularly from the performance standpoint. Often, however, the performance offered by blockchain frameworks is insufficient due to the complexity of the state machine replication process. In such cases, various performance/scaling techniques can be exploited to customize blockchain frameworks and obtain some performance improvements, albeit often at the cost of security or decentralization. In this chapter, we discuss how we successfully integrated parallel transaction execution in the Cosmos blockchain framework, a feat that poses both theoretical and practical challenges. The contents of this chapter are based on Ref. [27, 42, 41].

5.1 Introduction

Logistic processes often generate vast quantities of real-time data, necessitating robust processing capabilities. Thus, the successful implementation of blockchain in industrial settings hinges on ensuring optimal performance levels. However, within blockchain networks, all peers process the same operations in the same sequence, in accordance with the state machine replication model [185, 123]. Thus, workloads are replicated rather than distributed among the peers. In addition, some blockchain systems process transactions in sequence to avoid creating state inconsistencies among different peers, at the cost of constraining performance substantially.

Although multiple works deal with parallel transaction execution in the context of state machine replication (e.g., [138, 17]), traditional strategies may not be directly

applicable to blockchain systems. Blockchains often represent their state by leveraging hierarchical data structures, like Merkelized trees, instead of flat key-value stores. Merkelized trees allow for efficient data integrity verification, even when retrieved from potentially untrusted peers. Unfortunately, the use of Merkelized trees may impose stricter concurrency contraints due to the order-dependent nature of their operations. In particualr, some Merkelized trees introduce some indirect dependencies among transactions that are imposed by the contingent state of the tree structure, not by the transactions themselves. Taking these dependencies into account is mandatory to prevent the creation of multiple state representations and inconsistencies in blockchain systems. We may consider, for example, the inseriton of keys 1..4 in a Merkelized AVL tree, as shown in Figure 5.1. The transactions access different keys and should be concurrently executable under parallel state machine replication, but different transaction schedules may generate different trees. The two trees in Fig. 5.1 hold the same values in the same order but differ in their Merkle root hashes. We name this issue as the "ambiguous state representation problem".



(a) Insert order: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ (b) Insert order: $4 \rightarrow 3 \rightarrow 2 \rightarrow 1$

Fig. 5.1 Different insertion orders of the same set of values can result in different state representations in (Merkelized) AVL trees.

Apart from characterizing the ambiguous state representation problem, this chapter introduces an algorithm for parallel and optimistic transaction execution in blockchain systems that guarantees the creation of a consistent tree representation across all peers. Our approach is general and can be employed with any Merkelized tree structure. To the best of our knowledge, this is the first attempt to introduce parallelism into blockchains using such generalized data structures. The focal points of this chapter are as follows.

- Definition and formalization of the ambiguous state representation problem when introducing parallel transaction execution on Merkelized trees.
- Descrption of an algorithm for parallel transaction execution that avoids the ambiguous state representation problem.
- Practical implementation of our algorithm into the ppopular Cosmos SDK framework. Existing Cosmos applications only need to update a single package dependency to benefit from parallel transaction execution.
- Comprehensive performance evaluation with varying network configurations, based on standard workloads generated with the YCSB benchmark.

The subsequent sections of this chapter are structured as follows: Section 5.2 presents the ambiguous state representation problem and an overview of the Cosmos blockchain. Section 5.3 delineates our algorithm for avoiding the ambiguous state representation problem in parallel transaction execution. Section 5.4 presents a coprehensive performance evaluation of our algorithm under various network configurations. Section 5.5 discusses blockchain performance enhancement in our electric vehicle supply chain use case. Finally, Section 5.6 concludes the chapter.

5.2 Background

This section presents the ambiguous state representation problem and provides a brief introduction to the Cosmos blockchain, which we used to test our algorithm.

5.2.1 Problem description

We define a blockchain $B_{[h]}$ with the last committed block h for a set of peers $\mathscr{P} = \{0, 1, \ldots, (n-1)\}$ in the following manner:

$$B_{[h]} := \{\sigma_{[h],0}, \sigma_{[h],1}, \ldots, \sigma_{[h],n-1}\}, \tag{5.1}$$

where $\sigma_{[h],i}$ is part of the set S comprising all Merkelized trees with values in the leaves representing a key-value store with keys in K. The function $MRoot : S \rightarrow \{0,1\}^{\kappa}$ generates the Merkle root of a Merkelized tree with security parameter κ.

The state transition function $f : S \times \Gamma \to S$ produces a new state tree $\sigma_{[h+1]} \in S$ by considering the current state $\sigma_{[h]} \in S$ and a schedule (i.e., a total order) Γ of transactions $T_{[h+1]} = \{t_0, t_1, \ldots, t_{m-1}\}$ committed in block $h+1$. Each transaction involves one or more read, insert, or delete operations on the state tree. These operations are atomic and target a single key/leaf in the state tree, with only insert and delete operations capable of modifying the tree.

The functions $Keys : T \to K$, $WriteSet : T \to K$, and $ReadSet : T \to K$ will be used in Section 5.3. *Keys* returns the set of keys targeted by a transaction, while $WriteSet$ and *ReadSet* return the mutative and read sets of a transaction, respectively. Notably, the keys in K solely identify the leaves of a tree. Therefore, transactions possess the same read/write sets if they operate on the same leaves, irrespective of inner nodes. Parallel transaction execution may result in different peers executing different schedules. For this reason, we introduce the following definitions.

Definition 5.2.1. Let tree $\sigma \in S$. Two schedules Γ_i and Γ_j of T are considered σ-equivalent, denoted as $\Gamma_i =_\sigma \Gamma_j$, if and only if:

$$MRoot(f(\sigma, \Gamma_i)) = MRoot(f(\sigma, \Gamma_j)). \tag{5.2}$$

Definition 5.2.2. $B_{[h]}$ is in a consistent state $\sigma_{[h]}$ if and only if:

$$\forall i, j \in \mathscr{P} : MRoot(\sigma_{[h],i}) = MRoot(\sigma_{[h],j}). \tag{5.3}$$

Theorem 5.2.1. Let $B_{[h]}$ be in a consistent state $\sigma_{[h]}$. $B_{[h+1]}$ is in a consistent state $\sigma_{[h+1]}$ if and only if $\forall i, j \in \mathscr{P} : \Gamma_i =_{\sigma_{[h]}} \Gamma_j$.

Proof. The definition of state transition function for each peer is as follows:

$$\forall i \in \mathscr{P} : \sigma_{[h+1],i} = f(\sigma_{[h],i}, \Gamma_i). \tag{5.4}$$

If we apply the *MRoot* function to both sides of the previous equation, we obtain:

$$\forall i \in \mathscr{P} : MRoot(\sigma_{[h+1],i}) = MRoot(f(\sigma_{[h],i}, \Gamma_i)). \tag{5.5}$$

However, $\sigma_{[h]}$ is consistent: $\forall i \in \mathscr{P} : \sigma_{[h],i} = \sigma_{[h]}$. Additionally, finding hash collisions is unlikely. Thus:

$$\forall i \in \mathscr{P} : MRoot(\sigma_{[h+1],i}) = MRoot(f(\sigma_{[h]}, \Gamma_i)). \tag{5.6}$$

Sufficiency. If $\forall i, j \in \mathscr{P} : \Gamma_i =_{\sigma_{[h]}} \Gamma_j$, then, by definition:

$$\forall i, j \in \mathscr{P} : MRoot(f(\sigma_{[h]}, \Gamma_i)) = MRoot(f(\sigma_{[h]}, \Gamma_j)). \tag{5.7}$$

We can now use Eq. 5.6 on both sides to obtain:

$$\forall i, j \in \mathscr{P} : MRoot(\sigma_{[h+1],i}) = MRoot(\sigma_{[h+1],j}). \tag{5.8}$$

Thus, $\sigma_{[h+1]}$ is consistent.

Necessity. If $\exists i, j \in \mathscr{P} : \Gamma_i \neq_{\sigma_{[h]}} \Gamma_j$, then, by definition:

$$\exists i, j \in \mathscr{P} : MRoot(f(\sigma_{[h]}, \Gamma_i)) \neq MRoot(f(\sigma_{[h]}, \Gamma_j)). \tag{5.9}$$

We can now use Eq. 5.6 on both sides to obtain:

$$\exists i, j \in \mathscr{P} : MRoot(\sigma_{[h+1],i}) \neq MRoot(\sigma_{[h+1],j}). \tag{5.10}$$

Thus, $\sigma_{[h+1]}$ is not consistent.

□

The ambiguous state representation problem is introduced to address situations where different transaction schedules lead to distinct state representations due to indirect dependencies not visible at the transaction level. In fact, the Merkle root of a Merkle tree embeds information on both the state and the internal structure of the tree representing it. Thus, it may happen that different nodes reach the same state but build different trees, resulting in Merkle root hash mismatches and consensus failures. Additionally, the issue of transaction fees being paid to the block proposer often results in write conflicts among all transactions within a block. Our algorithm proposes a workaround for both these problems.

5.2.2 Cosmos

Cosmos [117] operates as a network of interconnected blockchains, using the Inter-Blockchain Communication protocol for cross-chain data transfer. The Cosmos ecosystem provides various tools for facilitating blockchain creation and network expansion. The software stack of a typical Cosmos blockchain, illustrated in Fig. 5.2,

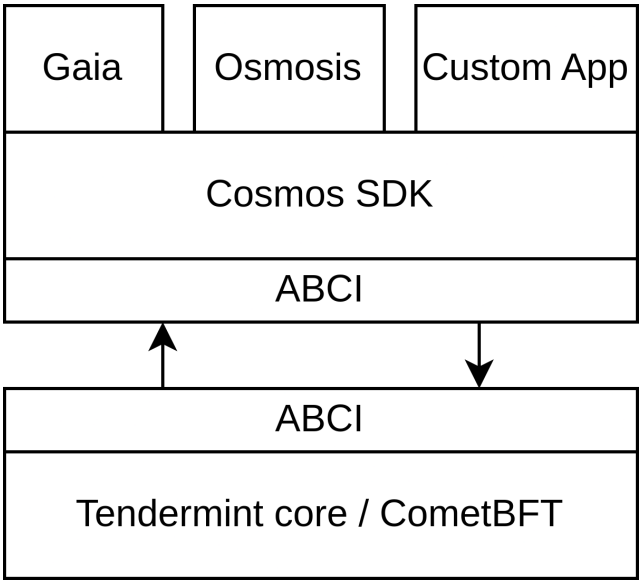


Fig. 5.2 Software stack of a Cosmos SDK application.

includes Tendermint Core (now Comet BFT) and the Cosmos SDK. Comet BFT is responsible for managing state machine replication while Cosmos SDK streamlines application-level development. The two modules interact through the ABCI interface. The following methods are invoked in the following order:

- BeginBlock: signals the start of a new block to the application.
- DeliverTx: delivers a single transaction to the application, with this method being invoked once per transaction.
- EndBlock: indicates the end of the block.
- Commit: instructs the application to produce the Merkle Root and return it to the Tendermint Core.

In the ABCI 2.0 specification, a change is made to the transaction processing methods, introducing a single method, FinalizeBlock, to deliver all transactions to the application in a single operation.

Developers can use the Cosmos SDK to accelerate blockchain creation, as it supports multi-asset public Proof-of-Stake (PoS) blockchains and permissioned Proof-of-Authority (PoA) blockchains. The SDK provides functionality for account management, gas measurement, fee payment, and staking to developers.

This chapter demonstrates our approach to enhancing the default transaction execution efficiency of the Cosmos SDK.

5.3 Algorithm description

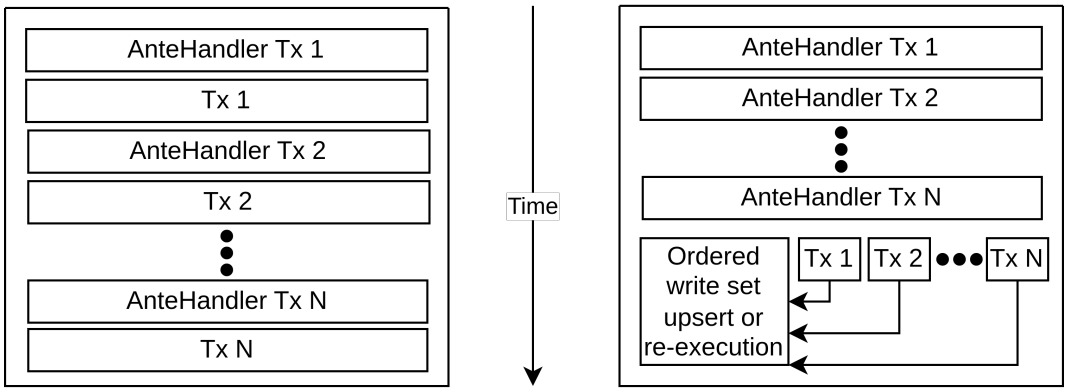


Fig. 5.3 Comparison of blockchain transaction execution: default (left) and our proposal (right).

Our analysis of the ambiguous state representation problem reveals that executing write operations out of order could lead to distinct tree representations, even when the operations target different tree leaves. However, these representations only differ by some rotation operations, implying that they assign the same values to the same keys. Consequently, while these representations cannot be used to compute the Merkle root hash, they remain useful for reading and writing values. In a broader context, peers can leverage flat key-value stores for reading and writing values, provided they can compute the same Merkle root hash in the end.

Our solution, as outlined in Algorithm 1, relies on several key functions:

- *Iterator (txs)*: generates an iterator for the given array of objects.
- *Next (i)*: advances iterator *i* to the next value.
- *HasNext (i)*: checks if the iterator has reached the end of the array.
- *GetValue (i)*: retrieves the value currently pointed to by iterator *i*.

- *AnteHandleSync (tx, store)*: executes the ante handler for transaction *tx* synchronously. Values are read from /written to *store*. It returns the updated store.
- *CreateArray ()*: returns an empty array.
- *WrapStore (store)*: creates a hashmap that acts as a write-back cache for *store*. A dirty bit is used to identify modified data, allowing the separation of read and write sets.
- *ExecuteAsync (tx, store)*: executes *tx* asynchronously by reading from and writing to *store*.
- *ExecuteSync (tx, store)*: executes *tx* synchronously by reading from and writing to *store*.
- *Append (array, elm)*: appends *elm* to the end of *array*, returning the newly created array.
- *PopFront (array)*: removes the first value of the array and returns it.
- *Wait (future)*: waits for *future* to complete.
- *IsRWConflict (firstCache, secondCache)*: checks if the read set of *secondCache* contains at least one value also present in the write set of *firstCache*.
- *WriteSetUpsert (firstCache, secondCache)*: merges *firstCache* and *secondCache*, with the values in *secondCache* overwriting those in *firstCache*.
- *WriteBack (cache, store)*: writes the values in *cache* back to *store*.

We can deconstruct our implementation into three steps to clarify how the algorithm operates.

5.3.1 Sequential step: reordering

Invalid transactions should not be executed within blockchains. Therefore, an *ante handler* is executed prior to each transaction, performing both stateless and stateful validity checks. These checks may include verifying signatures, checking block expiration, and more. Notably, an essential check involves the payment of

transaction fees to the block proposer, necessitating the sequential execution of ante handlers. Consequently, interleaving ante handlers and transactions prevents the parallel execution of transactions. We propose to process transactions in parallel after sequentially executing all ante handlers, as illustrated in Figure 5.3.

5.3.2 Parallel step: optimistic execution

Following the sequential execution of all ante handlers, transactions are processed in parallel. Each transaction is linked to a write-back cache and interacts with it for both reading from and writing data. Consequently, each transaction operates without awareness of the updates made by others, and its read and write sets can be determined based on the contents of the associated write-back cache. As a result, peers can execute transactions in any order.

5.3.3 Sequential step: conflict resolution

To avoid ambiguous state representations, we propagate the writes of various transactions to the underlying state tree in the sequential order in which transactions appear in the block. However, before implementing such changes, we must prevent computational anomalies and potentially re-execute conflicting transactions. We decide whether a transaction t_i must be re-executed based on the following observations:

- $ReadSet(t_0,\ldots,t_{i-1}) \cap ReadSet(t_i) \neq \emptyset$: No re-execution is required since the state remains unaltered.
- $ReadSet(t_0,\ldots,t_{i-1}) \cap WriteSet(t_i) \neq \emptyset$: As each transaction is unaware of modifications made by others, the values read by $t_0,\ldots,t_{i-1}$ remain unaffected by the writes of t_i. Therefore, no re-execution is necessary.
- $WriteSet(t_0,\ldots,t_{i-1}) \cap ReadSet(t_i) \neq \emptyset$: t_i needs to be re-executed as it did not observe the modifications made by its predecessors.
- $WriteSet(t_0,\ldots,t_{i-1}) \cap WriteSet(t_i) \neq \emptyset$: Two scenarios can be distinguished. If t_i carries out updates, a conflict between the read set of t_i and the write set of its predecessors also exists, leading back to the previous case. If t_i performs blind writes, its writes can be propagated to the underlying state tree without re-execution.

In essence, re-execution is necessary only if $WriteSet(t_0, \ldots, t_{i-1}) \cap ReadSet(t_i) \neq \emptyset$. In all other cases, the writes of t_i can be propagated to the underlying state tree.

5.4 Experiment analysis

Our integration of the algorithm within the Cosmos SDK was aimed at demonstrating the feasibility of our solution and evaluating potential performance improvements. To this end, we developed a key-value store application on top of the Cosmos SDK and compared the average number of committed Transactions Per Second (TPS) between the official release and our parallel implementation. Both versions used the same key-value store application, as a single dependency needs to be updated to switch between the official release and our implementation. Throughout our tests, we did not encounter any disruptions in consensus, affirming the determinism of our solution.

5.4.1 Setup and environment

We established a network of 40 Virtual Machines (VMs) on AWS, each equipped with the c6a.2xlarge instance family and 120 GB of gp3 SSD. Each VM hosted a Docker container in host network mode, and the tests were conducted using version 0.45.13-ics of the Cosmos SDK. The Yahoo! Cloud Serving Benchmark (YCSB) was employed to generate workloads, including the update-heavy, read-mostly, read-latest, and read-modify-write workloads (named A, B, D, and F respectively). The system performance was evaluated externally, with each client submitting 10^5 transactions to every peer in a round-robin fashion. We explored various network configurations, thread counts, and pre-existing record counts during our tests.

5.4.2 Performance analysis: thread count

Figure 5.4 depicts the performance variations with different thread counts. Increasing the number of threads led to overall performance improvements. Notably, the throughput reached a saturation point of 400 TPS with just two threads in workload F. Parallel transaction execution consistently outperformed the single-threaded system,

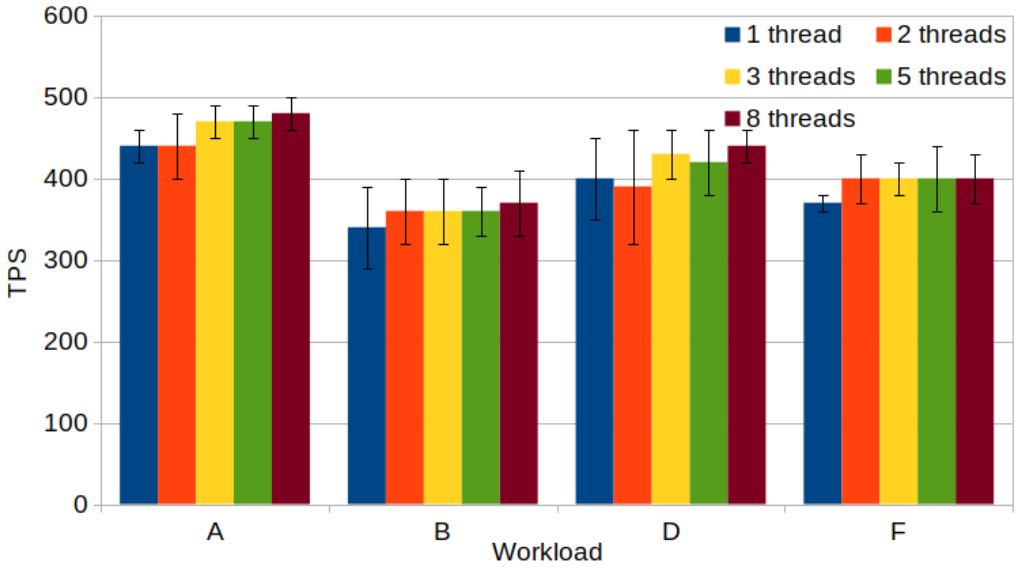


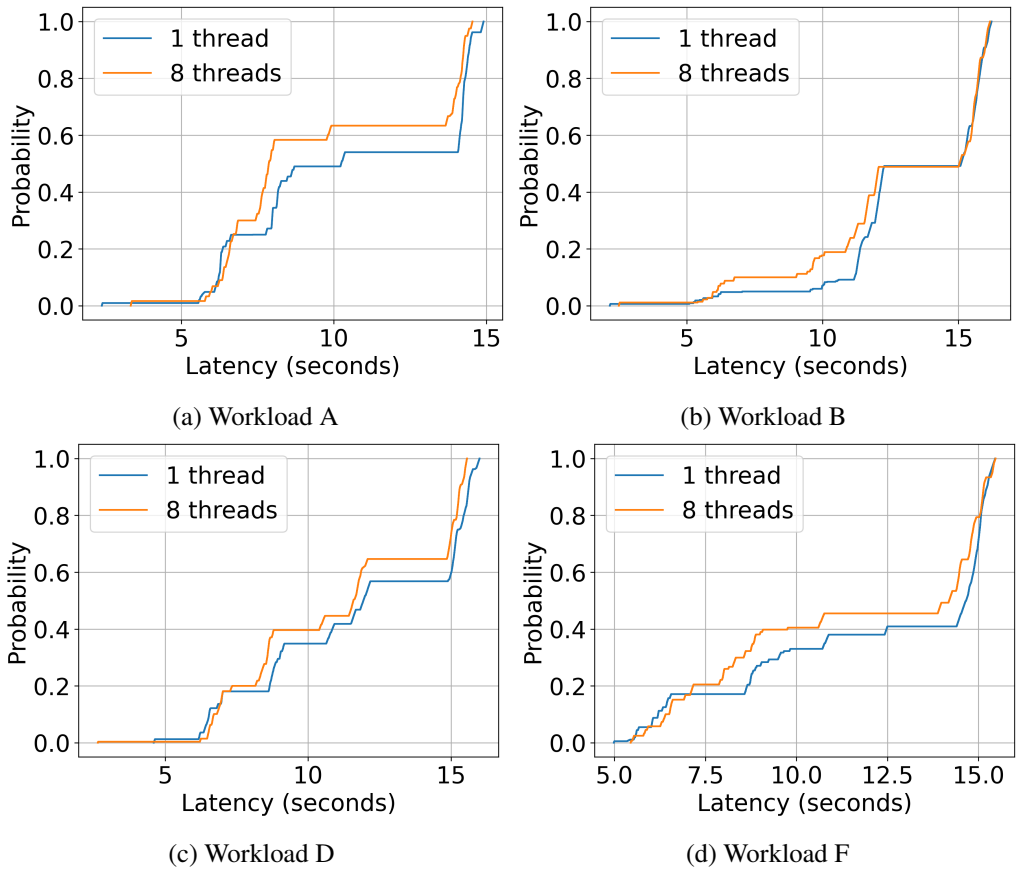
Fig. 5.4 Performance while varying numbers of threads.

providing higher throughput across all workloads, despite the moderate performance gains observed in workloads involving simple computations.

The latency cumulative distributions are illustrated in Figure 5.5 for the single-threaded and 8-threaded systems. The parallel approach consistently demonstrated superior performance over the single-threaded system in all workloads. For instance, in workload A, the multithreaded execution reduced the median latency from 10.3 seconds to 7.9 seconds.

5.4.3 Performance analysis: network size

Figure 5.6 showcases the performance trends with varying network sizes. With the exception of the 4-node network under workload B, parallel transaction execution consistently provided higher throughput. While no clear correlation was observed between the performance gains and the network size, the significance of the tests conducted with 40 nodes remains useful for various real-world deployment scenarios.



(a) Workload A

(b) Workload B

(c) Workload D

(d) Workload F

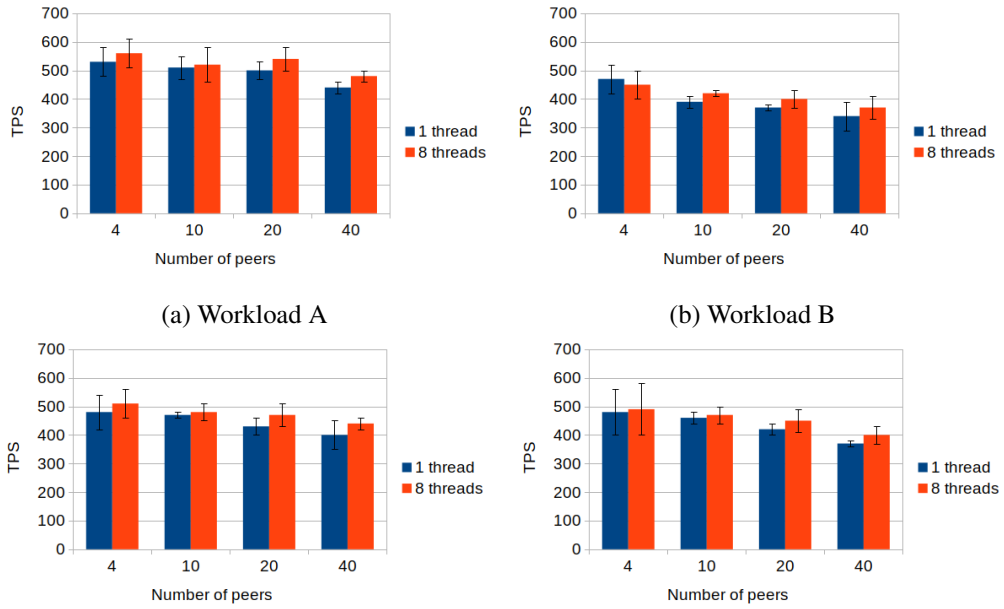
Fig. 5.5 Comparison of the latency cumulative distributions.

5.4.4 Performance analysis: database size

Figure 5.7 demonstrates the performance variations with different numbers of pre-existing records. Similar to the previous cases, parallel transaction execution consistently resulted in higher throughput, with increasing gains as the number of pre-existing records grew. This trend is rational, considering the increased traversal time for larger state trees. Future research may delve deeper into the impact of parallel transaction execution on even larger state trees.

5.4.5 Summary

To summarize our findings:



(a) Workload A

(b) Workload B

(c) Workload D

(d) Workload F

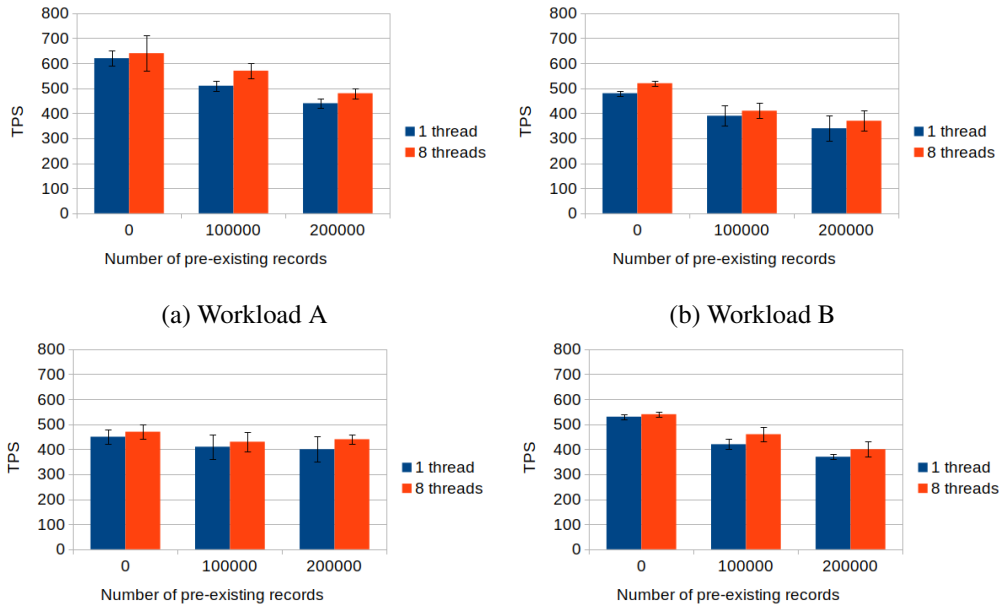
Fig. 5.6 Performance of different network sizes.

- Parallel transaction execution offers benefits in terms of throughput and latency across all workloads.
- The ambiguous state representation problem poses significant constraints on parallel transaction execution, resulting in reduced throughput and latency gains.

5.5 Use case: electic vehicle supply chain

Our electric vehicle supply chain use case necessitates the tracking of millions of batteries and vehicles. Therefore, achieving a performance level of tens of thousands of transactions per second is imperative for implementing a continuous monitoring approach. Based on the results of this chapter, simply relying on parallel transaction execution is insufficient to bridge the considerable performance gap spanning multiple orders of magnitude between the required and available performance capacities of blockchain frameworks.

As a result, we have chosen to abandon continuous monitoring and instead focus on storing information related to harmful events. This decision significantly reduces



(a) Workload A

(b) Workload B

(c) Workload D

(d) Workload F

Fig. 5.7 Performance while varying numbers of pre-existing records in the blockchain state.

the required throughput to a few dozen transactions per second. In this context, fine-tuning performance through parallel transaction execution is likely unnecessary. However, we may revisit this decision when transitioning to production, where we anticipate dealing with more complex transactions.

5.6 Conclusion

The ambiguous state representation problem can lead to inconsistencies in the state of a blockcahain when executing transactions in parallel. This chapter introduces an algorithm that addresses this issue by ensuring that write operations are carried out in a specific order, without fully compromising concurrency. We integrated our algorithm into the widely used Cosmos SDK to demonstrate its viability and examine the potential performance enhancements. We conducted a comprehensive performance assessment, indicating that our implementation consistently outperformed the existing one by 5%-10% in terms of Transactions Per Second (TPS) across all scenarios. Other techniques [221, 79] yield better results in terms of eficiency gains, but do not address the ambiguous state representation problem.

The performance gain we obtained falls short of resolving the scalability challenges inherent in blockchain technology. Nonetheless, our optimizations have yielded tangible benefits for end users, as validated through evaluations conducted from an external client, even within networks consisting of numerous peers.

After achieving sufficient performance, we may start defining the logic of our application, which translates to designing smart contracts in the blockchain context. In Chapter 6, we delve into this topic by discussing some guidelines for smart contract standardization.

Algorithm 1 ProcessBlock executes block transactions and updates the state tree coherently

```
1: function ProcessBlock(store, txs)
2:     i ← Iterator(txs) ▷
3:     repeat
4:         i ← Next(i) ▷
5:         tx ← GetValue(i) ▷
6:         store ← AnteHandleSync(tx, store) ▷
7:     until HasNext(i) ▷
8:     fwss ← CreateArray() ▷
9:     accum ← WrapStore(store) ▷ Wrapped stores are hashmaps acting as write-back caches
10:    i ← Iterator(txs) ▷
11:    repeat
12:        i ← Next(i) ▷
13:        tx ← GetValue(i) ▷
14:        ws ← WrapStore(store) ▷
15:        fws ← ExecuteAsync(tx, ws) ▷ fws is a future-like object
16:        fwss ← Append(fwss, fws) ▷
17:    until HasNext(i) ▷
18:    i ← Iterator(txs) ▷
19:    repeat
20:        i ← Next(i) ▷
21:        tx ← GetValue(i) ▷
22:        fws ← PopFront(fwss) ▷
23:        ws ← Wait(fws) ▷ Wait for future to be ready
24:        if IsRWConflict(accum, ws) then ▷ re-execute if read-write conflicts arise
25:            accum ← ExecuteSync(tx, accum) ▷
26:        else
27:            accum ← WriteSetUpsert(accum, ws) ▷ accum is updated with values in ws
28:        end if
29:    until HasNext(i) ▷
30:    WriteBack(accum, store) ▷ Writes are sequentially propagated to the underlying store, avoiding the ambiguous state representation problem
31:    return
32: end function
```

Chapter 6

Demystifying smart contracts

Blockchain-based applications use smart contracts to implement the application logic. The guiding principles for software design, development, testing, and validation in centralized systems seamlessly extend to smart contracts, with the only caveat that smart contracts must be deterministic and guarantee consistent executions across different peers at any time. Nonetheless, the actual benefits and limits of smart contracts are often misunderstood, and many misconceptions are widespread in the field. In this chapter, we address some of the most frequent ones and define guidelines for smart contract standardization. The contents of this chapter are based on Ref. [27, 37, 41].

6.1 Introduction

Digital tokens are experiencing increased adoption, reshaping business models as traditionally illiquid assets now find association with digital tokens and become tradable on decentralized finance marketplaces. In logistics, tokenization represents a way for companies to add transparency to the history of products, promote quality, and gain visibility over social- or environmental-friendly practices they employ. Tokenization is permeating various other sectors as well, with Non-Fungible Tokens (NFTs) and exchangeable coins gaining prominence in the market [148].

The pivotal drivers of this paradigm shift toward tokenization are smart contracts, which provide companies and individuals with flexibility, security, and automation [66]. These tamper-resistant computer programs leverage blockchain technology to

ensure the accuracy of their executions. Smart contracts find application in diverse objectives, such as ensuring data source quality, safeguarding assets and tokens, and augmenting and automating fairness in business processes. One particularly compelling objective involves the automation of legal contracts [80].

Nonetheless, creating tamper-resistant, decentralized, secure, legally recognized, and economically advantageous smart contracts is a complex endeavor that demands consideration of various technical and legal aspects [106]. Smart contracts seldom operate in isolation; for instance, in logistics, drones handle on-field operations, while IoT devices gather on-field data for analysis through artificial intelligence, stochastic programming, and granular computing approaches. Ensuring decentralized data feeds for smart contracts often requires tracking a single product using multiple IoT devices. However, this approach can be impractical due to economic or physical constraints, and processing data from redundant sources adds complexity to algorithms without necessarily enhancing accuracy. Establishing a comprehensive smart contract framework poses challenges as multiple concerns must be addressed concurrently, leading to the recent emergence of standards in the literature [167].

Despite the growing interest in smart contracts, numerous misconceptions persist. Defining smart contracts is challenging from a technical standpoint, given the varied functionalities offered by blockchain platforms, each employing different deployment and execution strategies. This diversity, coupled with the diverse backgrounds and priorities of those interested in the topic, has fostered the proliferation of misunderstandings and partial truths. The term "smart contract" itself can be misleading, suggesting legally recognized digital agreements rather than general-purpose computer programs. Furthermore, the properties of smart contracts often hinge on the configuration and decentralization level of the underlying blockchain system, making generalizations imprecise. In this context, the creation of precise definitions, standards, guidelines, and best practices for smart contracts has become a pressing necessity. Additionally, considering the broad audience interested in the topic, discussions about smart contracts must be conducted with precision, accuracy, and widespread understanding, adding an extra layer of complexity to the task.

This chapter aims to outline some basic principles defining smart contracts and define guidelines for future standards. The discussion may help researchers, lawyers, computer scientists, and decision-makers understand smart contracts' benefits and caveats. Our main objectives are as follows:

- We describe smart contracts in a multifaceted fashion to make them understandable by a broad audience.
- We identify, discuss, and correct some common smart contract misconceptions.
- We define some guidelines for implementing and executing smart contracts, highlighting the applicability, compromises, and limits of the technology. Hopefully, future standards will adopt our guidelines by adapting them to specific use cases, where principles must meet practicality.

The remainder of this chapter is structured as follows: Section 6.2 briefly introduces the problem addressed in this chapter; Section 6.3 discusses some guidelines for smart contracts; Section 6.4 focuses on smart contracts for our electric vehicle supply chain use case; Section 6.5 presents some final comments on smart contract standardization.

6.2 Problem statement

Presently, smart contract research is advancing rapidly across diverse domains. Researchers are challenged by communication due to their varied backgrounds and viewpoints, spanning computer science, economics, and law. Frequently, crucial nuances are overlooked to simplify descriptions of smart contracts, fostering an environment conducive to the proliferation of misconceptions and incomplete truths. This limitation even extends to emerging standards on smart contracts.

In this chapter, we aim to unravel prevalent misconceptions surrounding smart contracts, offering a high-level perspective on the subject. Our discussion navigates both technical and non-technical aspects without neglecting essential details. By doing so, we present guidelines intended to assist readers from diverse backgrounds in demystifying the technology, comprehending its limitations, and discerning potential areas of adoption.

Table 6.1 Categorization of this chapter's claims on smart contracts

Category	Claims: a smart contract...
Perspectives	is an equivalence class is a state-transition function is a computer program belongs to the system
Properties	is digital is tamper-resistant is likely bug-free if independently coded and verified
Misconceptions	has to be certified has to be stored on-chain is immutable might leave the state of the system unchanged has an intrinsic meaning and interpretation has legal value
Guidelines	should be declarative (not procedural) should be independently coded (preferably) should be independently tested (preferably) should be independently executed (preferably) may leverage validity proofs should not rely on local information (e.g., the node's clock) should avoid oracles as much as possible should avoid undefined behaviors (e.g., iteration order) should be inspectable and provide access to its source code should be integrated into a framework that defines its meaning may be integrated into a framework for legal recognition should be avoided if the code must be certified
Requirements	must be deterministic must be verifiable must store its output on-chain

6.3 Smart contracts: misconceptions and guidelines

This section discussess common misconceptions related to blockchain-based smart contracts and proposes some guidelines for their standardization. The content of this section are summarized in Table 6.1.

6.3.1 Smart contracts are state-transition functions

A conceptual lens for understanding a blockchain system involves framing it as a finite state machine [30]. In this conceptualization, smart contracts are state-transition functions, transitioning the system from one state to another. Specifically, a smart contract can be articulated as a function $\delta \colon \mathscr{S} \times \mathscr{I} \to \mathscr{S}$, where $\mathscr{S}$ denotes the finite and non-empty set of the blockchain's states and $\mathscr{I}$ represents the set of potential input transactions.

In a broader context, this vision emphasizes that smart contracts manipulate two distinct types of data: the (internal) ledger data, deemed trustworthy, and the (external) transaction data, whose accuracy demands validation. Depending on the use case, relying solely on ledger data may prove insufficient for verifying transaction data. Consequently, there exists an inherent constraint on the utility of smart contracts. While the option to overcome this constraint by accepting partially-verified transaction data exists, such an approach introduces the "garbage in, garbage out" dilemma and compromises the reliability of ledger data.

6.3.2 Smart contracts must alter the state of the system

In contrast with generic state-transition functions, smart contracts must modify the system's state; reading or elaborating data without modifying the state makes checking the execution correctness of the smart contracts impossible. For example, the following code snippet shows how to define pure or view functions [49] in Ethereum.

```
contract Logger {
    uint64 rowID;
    function persist(uint64 id) external {
```

```
        rowID=id;
    }
    function lastID() view external
        returns (uint64) {
    return rowID;
    }
}
```

When invoking pure or view functions, a single node handles the execution and could provide a wrong response. This is expected: view or pure functions are not smart contracts but commodity tools for retrieving data from the Ethereum blockchain or elaborating transaction data. Assimilating pure or view functions to smart contracts is a common misconception [178, 208] and is likely induced by their declaration occurring within a data structure labeled with the "contract" keyword. Not all frameworks have reserved keywords for marking functions that do not alter the ledger, but the general concept still applies: the only smart contract's verifiable output is the one stored on-chain. This happens because peers reach consensus on state transitions, which are a consequence of write operations, not read ones: malicious peers could answer queries even when not allowed to by only leveraging their copy of the ledger. However, they cannot do the same with write operations, which would require altering also the copies of honest peers.

This guideline has important implications from a practical standpoint. For example, when generating a document, the smart contract should also store on-chain the document or a fingerprint of it. Additionally, reading the document (or other data) may require querying multiple peers or requesting a single message digitally signed by all of them. This happens, for example, with platforms like Hyperledger Fabric that use flat key-value stores to represent their state. Alternatively, platforms like Ethereum leverage Merkle trees which, as discussed in Chapter 5, allow verifying data correctness by only knowing the Merkle root, even if such data is retrieved from a single malicious peer. This approach is more efficient, even though the Merkle root must still be queried to multiple peers. Thus, in general, data retrieval requires executing multiple queries. The impracticality of this approach often translates to users trusting a few reputable peers, introducing a degree of trust and centralization even in blockchain systems.

6.3.3 Smart contracts must be verifiable

Ensuring the verifiability of a smart contract's execution is crucial to prevent system forks and ensure consensus. However, verification time may differ significantly from the contract's execution time, posing limitations on the use of time-related primitives [1]. Even if all nodes share a common atomic clock, it only synchronizes contract execution, not verification. While this doesn't prohibit the use of time-related primitives, their incorporation requires careful consideration. For instance, checking if the timestamps of two blocks are less than a month apart yields consistent results over time. Conversely, evaluating the same check with the last block's timestamp is problematic, leading to changing outcomes over time. Thus, time-based smart contracts, as proposed in Ref. [167], may introduce inconsistencies during verification. Therefore, smart contracts should solely rely on input transaction information or blockchain-stored data, excluding local factors like the executing node's time measurement.

This guideline holds significant implications in managerial and legal contexts, as certain use cases may be challenging to model through smart contracts. In blockchain systems, determining if an event occurred within a specific time window is feasible, but pinpointing the exact moment is not. In Bitcoin [124], for instance, block validity checks implement measures to prevent miners from manipulating block timestamps, but their accuracy is limited to a few hours. Consequently, deadlines can only be approximately verified.

6.3.4 Smart contracts must be deterministic

The preceding section underscores a broader fundamental aspect. A smart contract should yield identical outputs across all executing nodes, not solely at a specific future moment. Put differently, smart contracts must exhibit determinism. This imperative is frequently overlooked and extends beyond time-based primitives. For instance, in the Go programming language, maps represent unordered sets of key-value pairs, making the iteration order unpredictable. Such behavior can impact serialization libraries. More broadly, the handling of randomness in smart contracts demands careful consideration. This issue is actively under investigation, with key techniques for its resolution grounded in multi-party computation [70].

Adopting a more expansive viewpoint, smart contracts necessitate the standardization of data representation and processing. This standardization ensures uniformity among different peers, enabling them to converge on the same state.

6.3.5 Smart contracts are equivalence classes

A prevalent misunderstanding revolves around considering a smart contract as a singular entity [89]. However, in blockchain systems, nodes lack the capability to verify the sequence of operations performed by others. Their evaluation is based on whether they arrive at the same state post-computation. Consequently, distinct transition functions yielding an identical resultant state are indistinguishable. Therefore, a smart contract represents a category of equivalent state transition functions. These functions, when provided with the same input state and symbol, generate identical output states. Illustrated below is a code snippet exemplifying two equivalent implementations of the same smart contract.

```
func sum(a uint8, b uint8) {
  a = a + b
  store(a)
}
func sum(a uint8, b uint8) {
  var i uint8
  for i = 0; i < b; i++ {
    a = a + 1
  }
  store(a)
}
```

Notably, a blockchain system operates seamlessly even when some nodes employ the first function, while others opt for the second, as evidenced by empirical verification on Github [36]. This observation gains significance when smart contracts require legal validity. In such cases, defining the actions smart contracts should perform becomes imperative, rather than specifying how they should execute [84]. Consequently, smart contracts are recommended to be articulated in declarative languages. The development of such languages is an active area of research, holding

the promise of rendering smart contracts more accessible for implementation and comprehension [84].

6.3.6 Smart contracts do not need to be stored on-chain

Another prevailing misconception, potentially influenced by the Ethereum protocol, is the notion that smart contracts must be stored on-chain [176, 183]. Contrary to this belief, as discussed in the preceding section, the decentralized nature of blockchain systems precludes a node from scrutinizing the computations undertaken by its peers. Consequently, storing a smart contract's code on-chain doesn't compel uniform implementation across all nodes. Instead, it merely serves as a straightforward method to disseminate the smart contract code to every participant in the blockchain. Public blockchains like Ethereum leverage this approach to automatically update the pool of smart contracts on each node. In contrast, alternative platforms such as Sawtooth [155] diverge from this practice, choosing not to store the code on-chain. Instead, individual peers assume the responsibility of installing the requisite smart contracts on their respective nodes. The underlying principle is that nodes need not be aware of the code others execute but only the eventual state they reach. With the presumption of an honest majority, consensus on a common state is achieved. This guideline's empirical substantiation is accessible on Github [36].

Storing the code of a smart contract on-chain could be valuable in other contexts. For example, it could be important to track which version of a smart contract was running at a given point in the past for verifiability purposes. Moreover, the on-chain version could be the one that a tribunal should use in case of litigation. However, if the on-chain implementation is the only one having legal value, the peer that codes it could manipulate the behavior of the smart contract for selfish purposes (e.g., by hiding a backdoor). Even if the honest majority forks the system to restore its correct state, a tribunal could be forced to overrule the majority-based decision of the blockchain network and favor the legally recognized branch generated by the malicious peer. Thus, if the on-chain implementation is the only one having legal value, additional strategies must be implemented to guarantee its correctness (e.g., formal approval from the majority of peers).

Using declarative languages for coding smart contracts introduces additional challenges regarding where they can be stored and in which format. Given that

multiple implementations of the same smart contract may exist, with each node using only one, and since all possible implementations are not known a priori, it is impossible to store all of them on-chain. Thus, only the source code of the smart contract (in a declarative language) may be stored on-chain, while the machine code should be generated autonomously by each peer and stored off-chain.

6.3.7 Smart contracts are not immutable

This misconception is related to the previous one. If a blockchain is immutable, and a smart contract is stored on-chain, then the smart contract is immutable [189, 181, 114]. However, the term *immutable* is misleading. More correctly, a blockchain is append-only. Consequently, even if what is stored in a blockchain cannot be altered, a newer version of it can always be appended to the ledger. Thus, even when smart contracts are stored on-chain, they can be updated. One possible strategy for the Ethereum blockchain is described in Ref. [97]: the smart contract stores the address of another smart contract in one of its state variables. Whenever the original smart contract is invoked, it propagates the invocation to the smart contract at the stored address. The behavior of the original smart contract is modifiable by updating the stored address. Additionally, the peer majority can always decide to soft/hard fork the system to replace a given smart contract. Thus, Ethereum smart contracts are immutable only if they do not implement an update mechanism and the majority is unwilling to alter them. Thus, Ethereum smart contracts are tamper-resistant, more than immutable. Other platforms (e.g., EOSIO [125]) allow updating smart contracts by overwriting the old code with the new one [110].

While storing a smart contract's code on-chain presents advantages, such as facilitating version tracking for verifiability and legal considerations, it introduces complexities. For instance, reliance on the on-chain version for legal validity creates a single point of failure, enabling the coding peer to manipulate the smart contract's behavior for personal gain (e.g., concealing a backdoor). Even if the honest majority initiates a fork to rectify the system's state, a tribunal might be compelled to override the network's majority-based decision, favoring the legally recognized branch forged by the malicious peer. Therefore, if the on-chain implementation holds exclusive legal value, additional strategies are imperative to ensure its correctness, such as formal endorsement from the majority of peers.

6.3.8 Smart contracts are not legal contracts

Due to their nomenclature, blockchain-based smart contracts are commonly assimilated to actual contracts [197, 131, 153], which is a fallacy [66]: blockchain-based smart contracts fundamentally function as computer programs. This inherent nature expands their utility to a vast array of applications, extending well beyond the confines of a mere agreement [66]. While smart contracts transcend the scope of legal contracts, they typically omit specifications regarding their intended users, and users do not digitally signing the smart contracts themselves, but the transactions for interacting with them. Consequently, the status of smart contracts as legal contracts is not inherently assured, necessitating validation through existing legal frameworks. In specific jurisdictions, the definition of a legal contract may seamlessly encompass smart contracts, while in others, parallel legal contracts may be requisite to legitimize their status [69]. The potential for smart contracts to automate legal contracts, at least partially, is acknowledged [144]. However, this intersection of computer science and law remains an evolving research domain, demanding meticulous hybrid framework design [106, 126].

In the broader context, an independent blockchain cannot supplant existing contracts or serve as an indisputable evidentiary source in legal disputes. Although smart contracts can streamline data exchange among multiple entities, their legal validation is contingent on jurisdiction. Managers contemplating the adoption of blockchain technology should exercise caution, recognizing that the reduction of legal costs is not an automatic guarantee and necessitates thoughtful consideration of economic implications.

6.3.9 Smart contracts do not provide meaning

Smart contracts function as computer programs, processing sequences of bits and generating corresponding sequences of bits. However, they lack explicit details regarding the meaning and correct interpretation of the produced bit sequences. For instance, a smart contract might store the sequence *e*, *s*, *t*, *a*, *t*, *e* on-chain. While it may be assumed that this sequence represents the word *estate*, such an assumption cannot be guaranteed. The letters might be the outcome of a random extraction, necessitating separate interpretation. Moreover, the word *estate* carries different meanings in Italian and English, contributing to potential disparities in interpretation.

Consequently, smart contracts alone are insufficient; they require external standards to establish the encoding/decoding of data and guide data consumers in its interpretation. The formulation of rules for data interpretation cannot be naively stored on the blockchain, as it would lead to a circular problem.

To some extent, various blockchain protocols (e.g., Ethereum) implicitly define data encoding and decoding standards. However, due to their agnostic and general-purpose nature, these protocols lack sufficient details to ensure a definitive and meaningful interpretation of stored data. This deficiency hinders the legal legitimization of smart contracts. From a managerial viewpoint, implicit rules for data interpretation may suffice when smart contracts are employed solely for data sharing among multiple companies. However, for legal validity, these rules must be explicit or, in some manner, widely recognized and acknowledged. In such cases, smart contracts should be seamlessly integrated into a hybrid framework encompassing both computer science and law-related aspects.

6.3.10 Preferably, smart contracts should be independently coded and deployed

Diverging from the prevalent practice in many blockchain networks, such as Ethereum, smart contracts should not be singularly coded and deployed universally across all nodes within the system. This stands in contrast to the fundamental concept of blockchain, which operates on a premise where nodes inherently distrust each other. Consequently, a node should refrain from accepting the implementation provided by other untrusted nodes, opting instead to independently implement all smart contracts to ensure their correctness. As long as all honest nodes share a common understanding of how smart contracts should impact the system's state, the various independent implementations should fall within the same equivalence class. Consequently, all nodes can converge on the same state, even when utilizing distinct implementations of identical smart contracts.

Regrettably, the demand for each node to individually implement its smart contracts proves impractical in numerous scenarios. Only some use cases involve consortium blockchains where a limited number of actors require a few shared smart contracts, as seen in logistics applications [166, 34]. In such instances, each actor can feasibly code and deploy its implementation of the smart contract. However, in public

blockchains, nodes often engage with only a subset of available smart contracts, lacking the necessary expertise and resources to craft independent implementations. Requiring each node to independently code all smart contracts, even those unused, becomes overly burdensome in such situations. Nonetheless, malicious exploitation of a bug becomes significantly challenging with just a few independent and uniformly distributed implementations, as any potential bug would likely affect only a segment of the network. Thus, promoting the creation of multiple implementations and allowing peers to choose which one to install on their node could mitigate the risk of bug exploits in smart contracts.

From a managerial perspective, this guideline significantly impacts the economics of blockchain-based projects, redistributing development costs across various nodes rather than sharing them. Additionally, ensuring that all independent implementations align within the same equivalence class demands extra efforts. Notably, creators of widely adopted smart contracts could fund the development of independent implementations to mitigate the risk of bug exploits, akin to existing bug bounty programs.

6.3.11 Preferably, smart contracts should be independently audited and tested

Smart contracts ought to undergo independent auditing and testing by nodes before their utilization commences. Consequently, analogous considerations to those outlined in the previous guideline remain pertinent. Nonetheless, this guideline is likely to garner more widespread adoption, given that testing a smart contract should be comparatively more straightforward than coding it. Facilitating the testing process, it is considered a best practice to publish the source code of smart contracts, as this simplifies the auditing of the machine code.

From a pragmatic standpoint, nodes may face challenges in testing smart contracts due to the same resource and competency constraints outlined in the previous guideline, particularly within public blockchain networks. Consequently, especially when economic incentives for testing are lacking, smart contracts may not be as secure and trustworthy as commonly assumed.

6.3.12 Smart contracts may leverage validity proofs

In certain scenarios, the demand for independent executions of computationally-intensive smart contracts may be too demanding, and opting for a single execution with proof of its correctness could be preferable. Consider the problem of solving a Rubik's cube, for example. Finding a sequence of moves that restores the cube's original configuration is fairly challenging. However, if the moves are provided, it becomes simple to verify whether or not they constitute a solution. The following code snippet illustrates the two alternative approaches:

```
func rubik_heavy(problem Problem) {
  solution := solve(problem)
  if solution != nil {
    store(solution)
  }
}
func rubik_light(problem Problem,
                 moves []Move) {
  solution := verify(problem, moves)
  if solution != nil {
    store(solution)
  }
}
```

Remarkably, the lightweight approach still necessitates executing a portion of the protocol as a smart contract, specifically the solution verification and storage. Thus, smart contracts inherently have a complexity baseline, as the verification and storage steps must always be carried out in a decentralized manner. Nonetheless, many blockchain platforms are successfully incorporating validity proofs to enhance scalability. In this context, zero-knowledge proofs, such as those used in protocols like zkSync, prove particularly beneficial [92, 81].

6.3.13 Smart contracts do not need to be certified

Regardless of the use case, smart contracts need not undergo certification, as certification bodies act as trusted third parties, and blockchain endeavors to eliminate such

entities. While it is in the nodes' best interest to thoroughly test and audit their smart contract implementations (Almakhour et al., 2020), even utilizing external software testing services, a blockchain system should not rely solely on specific authorities for validation. Smart contract correctness should be guaranteed by the consensus among nodes to reach the same state. Ideally, each node should independently code and test its implementation based on the strategies they see fit. If a blockchain system considers external authorities for smart contract certification, alternative strategies are preferable:

- Transition to a centralized system controlled by the certification body. However, this poses a risk of allowing the certification body control over the smart contract through a potential backdoor.
- Utilize oracles, trusted third parties providing real-world data to the blockchain. While oracles are often necessary for obtaining real-world data, certifying smart contracts by having certification bodies run the code and store the result in the blockchain can be simpler and more efficient. Employing a decentralized oracle network (Breidenbach et al., 2021) can limit the influence of each oracle, allowing verification of results produced by various certification bodies. Nonetheless, this solution ties decentralization to the number of certification bodies rather than the number of nodes.

From a broader perspective, any form of centralization undermines the value of blockchain-based smart contracts. Decision-makers should carefully evaluate the decentralization potential of a system before opting for blockchain-based solutions. In some cases, the impracticality of decentralized solutions may lead to accepting centralized compromises, potentially defeating the original purpose of employing blockchain.

6.3.14 Preferably, smart contracts should not rely on oracles

Dependence on oracles poses challenges not limited to the process of code certification. Trusting data from oracles inherently introduces a trusted third party, compromising the system's decentralization [144]. Unfortunately, entirely eliminating oracles from a blockchain system is seldom feasible. Consequently, smart contracts should minimize reliance on oracle-provided data, with the implementation

of strategies aimed at deterring oracle misconduct. Oracle services, such as Chainlink [26], employ economic incentives and multiple data feeds to mitigate manipulation attempts.

In a broader context, smart contracts do not resolve the "garbage in, garbage out" issue when oracles are in play. Therefore, because blockchain can prevent retroactive data manipulations and rarely proactive ones, decision-makers should assess whether such guarantees align with their specific use case and justify the adoption of the technology.

6.3.15 Smart contracts are (likely) bug-free

Assuming that multiple nodes independently code a smart contract, the likelihood of all implementations sharing the same bug diminishes. Nonetheless, it's plausible that these implementations might rely on a common library, and bugs impacting the library would consequently affect all implementations. Still, the underlying concept of the scalability trilemma [6] remains relevant: augmenting the number of independent implementations raises the coding effort and enhances the likelihood of attaining a bug-free smart contract, as discussed in Section 6.3.10.

It's worth emphasizing that achieving similarly bug-free programs in centralized settings would demand comparable implementation efforts. Given that such an approach deviates from standard industrial practices, its cost-effectiveness remains questionable. However, in the context of smart contracts managing valuable assets (e.g., decentralized finance protocols), this strategy could be considered to mitigate the risk of cyberattacks.

6.3.16 Smart contracts are (likely) tamper-resistant

The independent execution and verification by multiple nodes make smart contracts tamper-resistant, rendering the corruption of their execution reasonably unfeasible. However, this may not hold true if collusion incentives and opportunities for nodes exist in the system. Additionally, it's essential to note that tamper resistance doesn't inherently ensure that a smart contract behaves as expected. The resistance stems from multiple independent executions, while correctness is a result of diverse and

independent implementations. Even if all nodes implement identical versions, the smart contract retains tamper resistance (albeit not guaranteed to be bug-free).

Highlighting that the level of tamper resistance in smart contracts correlates with the decentralization degree of the blockchain network is crucial. The effectiveness of smart contracts wanes if a single node can exert influence or if nodes possess strong collusion motivations. Therefore, decision-makers must scrutinize the inter-node relationships before joining a blockchain network. In a broader sense, blockchain doesn't entirely eradicate trust issues, as peers still rely on the assumption that the majority behaves honestly.

6.3.17 Smart contracts belong to the system

The issue of ownership concerning smart contracts remains a contentious subject. Typically, the creator assumes the role of the owner, but there's flexibility for a smart contract to confer special privileges to a designated entity. However, it's vital to recognize that the management and execution of a smart contract rest with the nodes of the blockchain system. In the event that a majority of nodes decide to modify the smart contract, the nominal owner would wield no authority or entitlement to impede the changes. Consequently, the effective owner of a smart contract is, in essence, the blockchain system itself. Although this statement holds philosophical implications with potentially limited practical ramifications, it underscores the altered connotations that the concept of ownership assumes within a blockchain system.

6.4 Use case: electric vehicle supply chain

We applied the guidelines discussed in this chapter when designing the smart contracts for our electric vehicle use case. Even though the legal recognition of smart contracts is still a debated topic, we believe that a blockchain-based solution could act as a deterrent and favor the honest behaviors of the actors in the supply chain. In the end, the difficulties in replacing partners should push companies to establish long-term cooperation, and as long as the assignment process is fair and transparent, they should be willing to accept their responsibilities.

We decided to avoid storing personal data on the chain to circumvent regulatory issues, such as those related to the General Data Protection Regulation. Overall, we chose to keep smart contracts as lightweight as possible and move data off-chain where feasible. To this extent, we moved processing logic off-chain and treated smart contracts as data loggers that only check permissions and write data to Besu's event storage instead of contract storage. This prevents smart contracts from accessing past data but enhances efficiency [52].

In Chapter 7, we will extend this discussion by providing an overview of the complete solution.

6.5 Conclusion

In this chapter, we discussed some general guidelines for smart contract standardization. What emerges is that the future adoption of smart contracts is hard to predict: full decentralization is impractical and imposes many additional challenges, but accepting too many compromises for practicality risks defeating the purpose of leveraging smart contracts in the first place. Nonetheless, we are certain that standardization is key for the future of smart contracts, and, in our opinion, future standards should avoid treating smart contracts as a standalone technology but should contextualize them within broad-scope frameworks, taking into account economic, legal, and technical issues alike.

In Chapter 7, we will discuss the integration of IoT devices in blockchain systems and finalize the design of our blockchain-based system for our electric vehicle supply chain use case.

Chapter 7

Integrating oracles

The last missing piece to build a complete system for our electric vehicle supply chain use case is the integration of IoT devices tracking batteries and vehicles. In this chapter, we discuss the adoption of the Narrowband-IoT protocol to certify the origin of IoT data and reduce the risk of data manipulation. We then present the complete blockchain-based solution for our use case.

7.1 Introduction

Data sharing and digitalization are the key drivers of the Logistics 4.0 revolution. In the course of this thesis, we analyzed data sharing in-depth by discussing how blockchain technology enables the creation of decentralized databases where companies can share data without incurring the risk of unilateral or unauthorized modification attempts. In this chapter, instead, we focus on digitalization, which requires the creation of cyber-physical systems.

Cyber-physical systems are automated and self-adapting systems that bridge the gap between the physical and digital worlds, resulting from the integration of multiple technologies. In particular, IoT devices are responsible for locating and identifying physical entities and collecting vast amounts of data about them. Data mining and machine learning techniques help extract relevant information from the gathered data and react to potential issues within the supply chain. The Internet of Services (IoS) paradigm enables the interconnectivity and exchange of services among devices and humans.

Clearly, IoT devices are a key element of cyber-physical systems, serving as the interface between the physical and digital worlds. Fortunately, in recent years, the size, imprecision, and cost of IoT devices have drastically decreased, boosting their widespread adoption and improving the quality of sensed data.

Nonetheless, IoT integration in decentralized contexts remains tricky: the need for combining the digital and physical worlds is not contemplated in traditional blockchain networks like Bitcoin and adds an additional layer of complexity to logistic use cases. Sensors are responsible for the GIGO problem in blockchain systems, can be easily manipulated, and, for this reason, are one of the favorite targets of cyberattacks. Even more critically, just displacing a sensor may be sufficient to jeopardize a blockchain-based solution.

We address the problem of IoT integration in logistics networks by covering the following aspects:

- We propose to employ patented IoT devices [20] to reduce the attack surface on external data sources. The devices can leverage the Internet Services Provider's (ISP) network infrastructure to enrich sensed data with additional information, including time and position of the transmitting antenna.
- We finalize our system for the electric vehicle supply chain use case.

The remainder of this chapter is organized as follows: Section 7.2 discusses the GIGO problem, Section 7.3 analyzes the use of the Narrowband-IoT protocol for secure IoT communication, Section 7.4 describes the final architecture of our electric vehicle use case, and Section 7.5 concludes this chapter.

7.2 Problem statement

The GIGO problem is a well-known issue in computer science, referring to the idea that even correct processes will yield incorrect results if the input data is inaccurate. Data sources may be unreliable due to malicious behavior or unforeseen issues, which can arise from both trust and technical problems.

The issue of trust among companies has already been extensively analyzed in the literature from various perspectives, including inter-organizational systems, game

theory, and supply chain management. In particular, the Prisoner's Dilemma is often used to model relationships between companies [113]. In such a game, cooperation (i.e., trusting and being trustworthy) is the best strategy for the system as a whole, whereas defection is the best strategy for individual players. Interestingly, in a single iteration of the game, rational players are likely to defect based on the likelihood of encountering defecting players. However, in multiple iterations, cooperation emerges as the best strategy for all but the final round (as the game ends, there is no future incentive for cooperation) [32]. Consequently, while trust naturally emerges in the long run [24], a business crisis may push a company to defect, as immediate gains are valued more than future benefits [63]. For similar reasons, trust management systems and contractual solutions fail to ensure accountability when too little is at stake [72, 71]. Cooperation is sustainable only if its long-term value outweighs the short-term benefits of defection [14].

Trust is a prerequisite for the successful creation of inter-organizational systems [104, 218]. However, even when companies do not trust each other, blockchain can serve as a technological solution to trust issues. Unfortunately, the GIGO problem also extends to blockchain-based systems, which are often considered secure due to their decentralized nature, heavy use of cryptography, and high levels of data and execution redundancy. Specifically, oracles provide blockchain systems with unverifiable external data that cannot be obtained otherwise. While this extends the applicability of blockchain systems far beyond the exchange of digital assets, it also introduces a single point of failure that can jeopardize the entire system, as information asymmetry favors opportunistic behavior [212]. Given the impossibility of completely eliminating the dependency of logistics information systems on monitoring devices, we propose a technical solution that limits the exploitability of such devices.

7.3 Narrowband-IoT for enhanced connectivity

To the best of our knowledge, solving the GIGO problem is not possible from a technical standpoint, but some strategies may reduce the risk of processing garbage data. In particular, it is possible to employ special devices [20] that leverage the Narrowband-IoT protocol and the network infrastructure of the Internet Service

Provider to enrich sensor data with additional metadata retrieved from the antenna used for transmission. The main benefits of adopting such devices are as follows.

- The device is onboarded with a SIM card equipped with a cryptographic module that can handle asymmetric cryptography. The SIM's cryptographic module guarantees that the private key is randomly generated and only known to the SIM itself.
- The SIM directly communicates with the ISP's antennas through the Narrowband-IoT protocol. Thus, there is no need to introduce message brokers, as short-ranged protocols do (e.g., Bluetooth Low Energy).
- The SIM enriches the sensor's measurements with additional metadata like the position of the antenna used for transmission and the transmission time. Cross-checking SIM and sensor data allows the detection of manipulation attempts.
- The software on the SIM is digitally signed, and its integrity can be verified.

In addition to the above, Narrowband-IoT allows for connecting tens of thousands of devices to the same antenna, improves transmission in hardly reachable places, and is very energy-efficient. IoT devices using it may last more than ten years [173].

The main limitations of this solution relate to physical attacks on the sensors and collusion attempts with the ISP. Discouraging physical attacks may be achieved by making the cost of performing the attack exceed the value of the tracked product. This should be sufficient to prevent data manipulation attempts from supply chain companies, but may not always be feasible. Collusion attempts with the ISP are possible but unlikely. The ISP has no real advantage in jeopardizing its reputation and potentially losing many clients just to favor one of them. Nonetheless, relying on the ISP introduces a certain degree of centralization, and setting up countermeasures is recommended.

7.4 Use case: electric vehicle supply chain

In this section, we describe the complete arcitecture we designed for our use case.

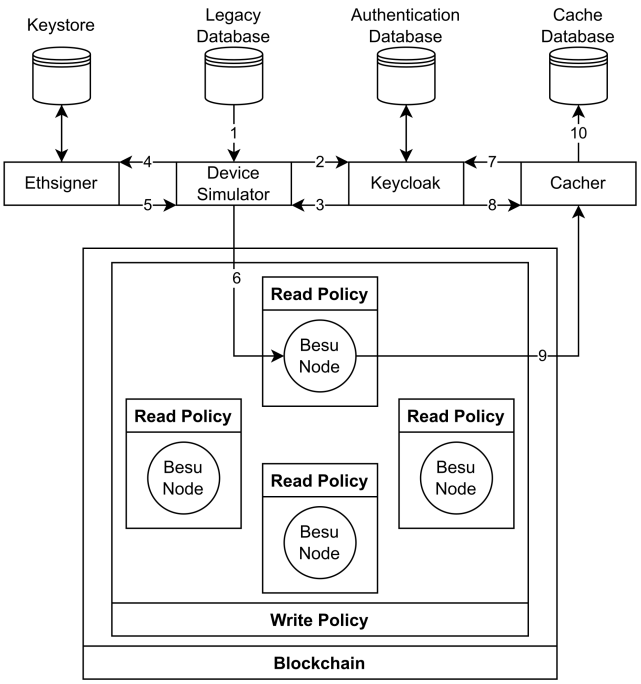


Fig. 7.1 Architecture of our solution.

7.4.1 System architecture

Blockchain adoption is challenging if blockchain-based solutions cannot process all the data required by production systems. For this reason, we decided to implement many of the features that would be present in a production environment and used some production data to evaluate the performance of our system to obtain meaningful results. In particular, we created a blockchain network of four nodes using the IBFT 2.0 consensus algorithm. The data stored in the blockchain is also persisted in an external database, namely the cache database. The cache database solves two issues: legacy application integration and query offloading. On-chain data is not encrypted, as data sharing is the objective of adopting blockchain technology.

Besu offers three layers of permissioning: node permissioning, account permissioning, and API permissioning. Node permissioning regulates peering. Only connections to other nodes of our system are allowed. Account permissioning regulates which accounts can update the ledger. We generated and granted permissions to a few thousand accounts representing IoT devices. API permissioning authorizes selected devices to invoke a node's API methods. We used Keycloak and the

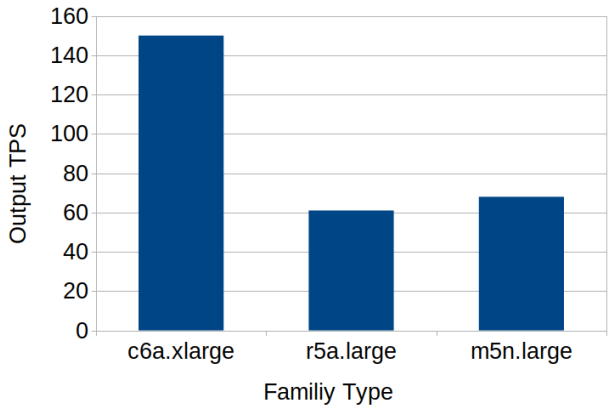


Fig. 7.2 Performance evaluation with multiple AWS instance families

OpenId Connect Client Credentials grant to restrict access to known clients. It is important to underline that account permissioning can be enforced at the network level, while node and API permissioning can only be enforced at the node level. Thus, Byzantine nodes cannot perform unauthenticated write operations but can answer unauthenticated read attempts. Such a limitation is common to all blockchain systems, which may limit the adoption of the technology. We used multiple clients to persist production data in the blockchain and Ethsigner to handle the private keys safely, enabling the integration with low-end IoT devices which may not be equipped with a cryptographic module. Nonetheless, such devices should be replaced by the ones that we previously described in this chapter. The proposed architecture is represented in Fig. 7.1.

7.4.2 Performance Evaluation

We tested our system on different AWS instance families to evaluate the impact of various factors, including the RAM size, the number of CPUs, and the network bandwidth. We used Hyperledger Besu v21.1.0 on all four nodes, as versions v22.7.4 and v22.7.6 turned out to be unreliable due to frequent crashes. We connected a single client to each node. A coordinator service instructs the clients on the workload to submit and synchronizes their interaction with the blockchain. We used workloads of 1600 transactions (400 per client) that we submitted to the blockchain at maximum input rate. We measured the number of transactions processed per

second (TPS) from when the coordinator submits a workload to the clients to when the last client completes the task. We used three types of instance families: AWS r5a.large instances are memory-optimized, AWS m5n.large instances are general purpose with large bandwidth, and AWS c6a.x large instances are compute optimized. The results of our performance evaluation are shown in Fig. 7.2.

AWS c6a.xlarge instances offer the best performance: such instances are equipped with four vCPUs, which is twice as much as the other two instance families. AWS m5n.large instances have slightly better performance than r5a.large instances, but we believe such a difference is a consequence of the different types of processors leveraged by the two families. Thus, the additional network bandwidth offered by m5n.large and the additional RAM provided by r5a.large instances may not improve the performance of our blockchain system. This result indicates that the bottleneck of our system is the number of vCPUs in the current configuration.

7.5 Conclusion

IoT devices acting as oracles are one of the most critical aspects of blockchain systems. On one side, they play a crucial role in interconnecting the physical world and the digital one; on the other, the data they provide is rarely verifiable and may be exploited to render blockchain systems unusable. Unfortunately, oracle data verifiability is still an open research issue, and there is no clear solution to it. At the moment, mitigation strategies are our best option. In this chapter, we discussed one possible mitigation strategy based on the adoption of specific IoT devices that can leverage the ISP network infrastructure to certify (to some extent) the origin of sensor data. We then finalized the design of the blockchain-based system for our electric vehicle supply chain use case.

In Chapter 8, we will provide some conclusive remarks on the main takeaways of this thesis.

Chapter 8

Conclusion

Supply chains are undergoing a revolution based on data sharing and digitalization, with blockchain standing out as a key enabler of this transformative process.

In this thesis, we delved into the adoption of blockchain in logistics networks, focusing on an electric vehicle supply chain use case. Our objective was to establish a single source of truth for supply chain companies, promoting fair and transparent assignment of responsibility, and ultimately reducing bureaucracy and litigations. Thus, we needed to design a system that could track information across the entire supply chain while ensuring the timely and secure storage and retrieval of gathered data.

We began by proposing a decision-making framework for blockchain adoption, which we applied to assess the feasibility of employing blockchain in our solution. Our conclusion affirmed that blockchain is well-suited for our use case but emphasized the necessity of integrating it with other technologies to establish trust in oracles.

Next, we conducted an analysis of popular blockchain frameworks to select the one aligning best with our goals. To compare framework performance, we developed a methodology aiming for consistent degrees of security and decentralization across all options. Ultimately, we chose Hyperledger Besu, considering its favorable compromise between performance, community support, features, resilience to malicious peers, and industrial readiness.

Recognizing the importance of optimizing performance for our solution, we investigated parallel transaction execution in blockchain frameworks. Addressing the ambiguous state representation problem in certain Merkle trees used in blockchains, we proposed an optimistic algorithm ensuring determinism without fully sacrificing parallelization. Although yielding some performance improvements, these were insufficient for our use case. Consequently, we decided to abandon continuous monitoring, opting to register only harmful events in our blockchain-based solution to reduce performance requirements.

Subsequently, we developed smart contracts to implement the application logic of our use case. Drawing from this experience, we formulated general and platform-agnostic guidelines for standardizing smart contracts, aiming to assist researchers and practitioners in their judicious use of smart contracts.

The final piece in designing our solution involved the integration of oracles. We advocated for the use of specialized IoT devices capable of enhancing sensor data with additional metadata based on the time and position of the ISP's antenna used for transmission. Cross-checking sensor and antenna data may help revealing potential data manipulation.

In conclusion, we believe our designed solution holds the promise of enabling supply chain companies to achieve the objectives of the use case, facilitating secure data sharing and reducing litigations. However, we acknowledge that the primary obstacles to blockchain adoption are not technical but managerial. The shift from industrial secrecy to data sharing and the integration of competition with cooperation must take place. Recognizing that achieving such a radical paradigm shift may take decades, we remain hopeful that, step by step, blockchain and tokenization will contribute to enhancing the social, environmental, and economic impact of supply chains.

References

[1] Manar Abdelhamid and Ghada Hassan. Blockchain and smart contracts. In *Proceedings of the 2019 8th International Conference on Software and Information Engineering*, page 91–95, 2019. ISBN 9781450361057. doi: 10.1145/3328833.3328857.

[2] Nicole Adarme. Tessera: The privacy manager of choice for consensys quorum networks, 2021. URL https://consensys.net/blog/quorum/tessera-the-privacy-manager-of-choice-for-consensys-quorum-networks/.

[3] E. Alchieri, F. Dotti, O.M. Mendizabal, and F. Pedone. Reconfiguring parallel state machine replication. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, volume 2017-September, pages 104–113, 2017. doi: 10.1109/SRDS.2017.23.

[4] E. Alchieri, F. Dotti, and F. Pedone. Early scheduling in parallel state machine replication. In *SoCC 2018 - Proceedings of the 2018 ACM Symposium on Cloud Computing*, pages 82–94, 2018. doi: 10.1145/3267809.3267825.

[5] T.A. Almeshal and A.A. Alhogail. Blockchain for businesses: A scoping review of suitability evaluations frameworks. *IEEE Access*, 9:155425–155442, 2021. doi: 10.1109/ACCESS.2021.3128608.

[6] A. Altarawneh, T. Herschberg, S. Medury, F. Kandah, and A. Skjellum. Buterin's scalability trilemma viewed through a state-change-based classification for common consensus algorithms. In *CCWC 2020*, pages 727–736, 2020. doi: 10.1109/CCWC47524.2020.9031204.

[7] M.J. Amiri, D. Agrawal, and A. El Abbadi. Parblockchain: 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.

[8] B. Ampel, M. Patton, and H. Chen. Performance modeling of hyperledger sawtooth blockchain. In *2019 IEEE International Conference on Intelligence and Security Informatics, ISI 2019*, pages 59–61, 2019. doi: 10.1109/ISI.2019.8823238.

[9] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018. ISBN 9781450355841. doi: 10.1145/3190508.3190538.

[10] J. Angelis and E. Ribeiro da Silva. Blockchain adoption: A value driver perspective. *Business Horizons*, 62(3):307–314, 2019. doi: 10.1016/j.bushor.2018.12.001.

[11] I.M. Ar, I. Erol, I. Peker, A.I. Ozdemir, T.D. Medeni, and I.T. Medeni. Evaluating the feasibility of blockchain in logistics operations: A decision framework. *Expert Systems with Applications*, 158, 2020. doi: 10.1016/j.eswa.2020.113543.

[12] A. Auinger and R. Riedl. Blockchain and trust: Refuting some widely-held misconceptions. In *International Conference on Information Systems 2018, ICIS 2018*, 2018.

[13] S. Baheti, P.S. Anjana, S. Peri, and Y. 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.

[14] George Baker, Robert Gibbons, and Kevin J Murphy. Relational contracts and the theory of the firm. *The Quarterly Journal of Economics*, 117(1):39–84, 2002.

[15] A. Baliga. Understanding blockchain consensus models, 2017. URL https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf.

[16] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. Performance evaluation of the quorum blockchain platform. *arXiv preprint arXiv:1809.03421*, 2018.

[17] E. Batista, E. Alchieri, F. Dotti, and F. Pedone. Early scheduling on steroids: Boosting parallel state machine replication. *Journal of Parallel and Distributed Computing*, 163:269–282, 2022. doi: 10.1016/j.jpdc.2022.02.001.

[18] Michele Battagliola, Andrea Flamini, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. Quadrans blockchain, 2021. URL https://quadrans.io/content/files/quadrans-yellow-paper-rev02.pdf.

[19] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia. A survey on blockchain interoperability: Past, present, and future trends. *ACM Computing Surveys*, 54(8), 2022. doi: 10.1145/3471140.

[20] Fabio Luigi Bellifemine, Danilo Gotta, and Tiziana Trucco. System and method for managing supply of electric energy through certified measures, may 2019.

[21] M. Belotti, N. Božić, G. Pujolle, and S. Secci. A vademecum on blockchain technologies: When, which, and how. *IEEE Communications Surveys and Tutorials*, 21(4):3796–3838, 2019. doi: 10.1109/COMST.2019.2928178.

[22] S. Benahmed, I. Pidikseev, R. Hussain, J. Lee, S.M.A. Kazmi, A. Oracevic, and F. Hussain. A comparative analysis of distributed ledger technologies for smart contract development. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, volume 2019-September, pages 1–6, 2019. doi: 10.1109/PIMRC.2019.8904256.

[23] M. Benedetti, F. De Sclavis, M. Favorito, G. Galano, S. Giammusso, A. Muci, and M. Nardelli. Certified byzantine consensus with confidential quorum for a bitcoin-derived permissioned dlt. In *CEUR Workshop Proceedings*, volume 3460, 2023.

[24] Simon Board. Relational contracts and the value of loyalty. *American Economic Review*, 101(7):3349–3367, 2011.

[25] W. Brammertz and A.I. Mendelowitz. From digital currencies to digital finance: the case for a smart financial contract standard. *Journal of Risk Finance*, 19(1):76–92, 2018. doi: 10.1108/JRF-02-2017-0025.

[26] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov1, Alexandru Topliceanu1, Florian Tramèr, and Fan Zhang. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. Technical report, Chainlink, 2021.

[27] Maria Elena Bruni, Vittorio Capocasale, Marco Costantino, Stefano Musso, and Guido Perboli. Decentralizing electric vehicle supply chains: Value proposition and system design. In *Proceedings - International Computer Software and Applications Conference*, volume 2023-June, page 1756 – 1761, 2023. doi: 10.1109/COMPSAC57700.2023.00271.

[28] Daniel Bumblauskas, Arti Mann, Brett Dugan, and Jacy Rittmer. A blockchain use case in food distribution: Do you know where your food has been? *International Journal of Information Management*, 52, 2020. ISSN 0268-4012. doi: 10.1016/j.ijinfomgt.2019.09.004.

[29] A. Burgos, E. Alchieri, F. Dotti, and F. Pedone. Exploiting concurrency in sharded parallel state machine replication. *IEEE Transactions on Parallel and Distributed Systems*, 33(9):2133–2147, 2022. doi: 10.1109/TPDS.2021.3135761.

[30] Vitalik Buterin. A next-generation smart contract and decentralized application platform. Technical report, Ethereum Foundation, 2014.

[31] Vitalik Buterin. On public and private blockchains, 2015. URL https://sawtooth.hyperledger.org/docs/core/releases/1.2.6/introduction.html.

[32] Marie-Laure Cabon-Dhersin and Shyama V Ramani. Opportunism, trust and cooperation: A game theoretic approach with heterogeneous agents. *Rationality and Society*, 19(2):203–228, 2007.

[33] G. Caldarelli. Understanding the blockchain oracle problem: A call for action. *Information (Switzerland)*, 11(11):1–19, 2020. doi: 10.3390/info11110509.

[34] V. Capocasale, D. Gotta, S. Musso, and G. Perboli. A blockchain, 5g and iot-based transaction management system for smart logistics: An hyperledger framework. In *COMPSAC 2021*, pages 1285–1290, 2021. doi: 10.1109/COMPSAC51774.2021.00179.

[35] Vittorio Capocasale. Blockchain adoption decision counselor, 2022. URL https://github.com/vittoriocapocasale/BlockchainAdoptionDecisionCounselor.

[36] Vittorio Capocasale. Smart contract equivalence, 2022. URL https://github.com/vittoriocapocasale/SmartContractEquivalence.

[37] Vittorio Capocasale and Guido Perboli. Standardizing smart contracts. *IEEE Access*, 10:91203 – 91212, 2022. doi: 10.1109/ACCESS.2022.3202550.

[38] Vittorio Capocasale and Guido Perboli. A decision framework for blockchain adoption, 2022.

[39] Vittorio Capocasale, Stefano Musso, and Guido Perboli. Interplanetary file system in logistic networks: a review. In *Proceedings - 2022 IEEE 46th Annual Computers, Software, and Applications Conference, COMPSAC 2022*, page 1684 – 1689, 2022. doi: 10.1109/COMPSAC54236.2022.00268.

[40] Vittorio Capocasale, Danilo Gotta, and Guido Perboli. Comparative analysis of permissioned blockchain frameworks for industrial applications. *Blockchain: Research and Applications*, 4(1), 2023. doi: 10.1016/j.bcra.2022.100113.

[41] Vittorio Capocasale, Maria Elena Bruni, and Guido Perboli. Technological insights on blockchain adoption: the electric vehicle supply chain use case. Unpublished, 2024.

[42] Vittorio Capocasale, Ferando Pedone, and Guido Perboli. Parallel transaction execution in blockchain and the ambiguous state representation problem. in press, 2024.

[43] M.P. Caro, M.S. Ali, M. Vecchio, and R. Giaffreda. Blockchain-based traceability in agri-food supply chain management: A practical implementation. In *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany, IOT Tuscany 2018*, pages 1–4, 2018. doi: 10.1109/IOT-TUSCANY.2018.8373021.

[44] B. Carson, G. Romanelli, P. Walsh, and A. Zhumaev. Blockchain beyond the hype: What is the strategic business value? *McKinsey Quarterly*, 2018(4): 118–127, 2018.

[45] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[46] Chainstack. Enterprise blockchain protocols evolution index 2020, 2020. URL https://chainstack.com/resources/#enterprise-blockchain-protocols-evolution-index-2020.

[47] Chainstack. Enterprise blockchain protocols evolution index, 2021. URL https://chainstack.com/download/enterprise-blockchain-protocols-evolution-index-2021/.

[48] David C Challener, Maria E Vachino, James P Howard II, Christina K Pikas, and Anil John. Blockchain basics and suitability: A primer for program managers. *Journal of Information Technology Management*, 30(3):33–44, 2019.

[49] P. Chapman, D. Xu, L. Deng, and Y. Xiong. Deviant: A mutation testing tool for solidity smart contracts. In *Blockchain 2019*, pages 319–324, 2019. doi: 10.1109/Blockchain.2019.00050.

[50] Z. Chen, X. Qi, X. Du, Z. Zhang, and C. Jin. Peep: A parallel execution engine for permissioned blockchain systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12683 LNCS:341–357, 2021. doi: 10.1007/978-3-030-73200-4_24.

[51] J. Chod, N. Trichakis, G. Tsoukalas, H. Aspegren, and M. Weber. On the financing benefits of supply chain transparency and blockchain adoption. *Management Science*, 66(10):4378–4396, 2020. doi: 10.1287/mnsc.2019.3434.

[52] Joseph Chow. A guide to events and logs in ethereum smart contracts, 2016. URL https://consensys.net/blog/developers/guide-to-events-and-logs-in-ethereum-smart-contracts/.

[53] Christopher D Clack, Vikram A Bakshi, and Lee Braine. Smart contract templates: essential requirements and design options. *arXiv preprint arXiv:1612.04496*, 2016.

[54] T. Clohessy, H. Treiblmaier, T. Acton, and N. Rogers. Antecedents of blockchain adoption: An integrative framework. *Strategic Change*, 29(5): 501–515, 2020. doi: 10.1002/jsc.2360.

[55] Come-from-Beyond. Decentralized vs distributed, or why dlt is (probably) an incorrect term, 2020. URL https://medium.com/@comefrombeyond/decentralized-vs-distributed-or-why-dlt-is-probably-an-incorrect-term-fccbf62bdfe7.

[56] CONCORDIA Consortium. CONCORDIA Web Site, 2016. URL https://www.concordia-h2020.eu/.

[57] ConsenSys. Scaling consensus for enterprise: Explaining the ibft algorithm, 2018. URL https://consensys.net/blog/enterprise-blockchain/scaling-consensus-for-enterprise-explaining-the-ibft-algorithm/.

[58] ConsenSys. Goquorum enterprise ethereum client, 2020. URL https://docs.goquorum.consensys.net/en/stable/.

[59] ConsenSys. Build on quorum, the complete open source blockchain platform for business, 2021. URL https://consensys.net/quorum/.

[60] Amie Corso. *Performance analysis of proof-of-elapsed-time (poet) consensus in the sawtooth blockchain framework*. PhD thesis, University of Oregon, 2019.

[61] T G Crainic, G Perboli, and M Rosano. Simulation of intermodal freight transportation systems: a taxonomy. *European Journal of Operational Research*, 270(2):401–418, 2018. doi: 10.1016/j.ejor.2017.11.061.

[62] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Recent advances in multi-dimensional packing problems. *New technologies-trends, innovations and research*, pages 91–110, 2012.

[63] Martin W Cripps, Eddie Dekel, and Wolfgang Pesendorfer. Reputation with equal discounting in repeated games with strictly conflicting interests. *Journal of Economic Theory*, 121(2):259–272, 2005.

[64] H. Cui, R. Gu, C. Liu, T. Chen, and J. Yang. Paxos made transparent. In *SOSP 2015 - Proceedings of the 25th ACM Symposium on Operating Systems Principles*, pages 105–120, 2015. doi: 10.1145/2815400.2815427.

[65] Michael del Castillo. Blockchain 50 2021, 2021. URL https://www.forbes.com/sites/michaeldelcastillo/2021/02/02/blockchain-50/.

[66] Marco Dell'Erba. Demystifying technology. do smart contracts require a new legal framework? regulatory fragmentation, self-regulation, public regulation., 2018. URL https://doi.org/10.2139/ssrn.3228445.

[67] T.T.A. Dinh, J. Wang, G. Chen, R. Liu, B.C. Ooi, and K.-L. Tan. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume Part F127746, pages 1085–1100, 2017. doi: 10.1145/3035918.3064033.

[68] Abhishek Dixit, Vipin Deval, Vimal Dwivedi, Alex Norta, and Dirk Draheim. Towards user-centered and legally relevant smart-contract development: A systematic literature review. *Journal of Industrial Information Integration*, 26: 100314, 2022.

[69] Mateja Djurovic and André Janssen. The formation of blockchain-based smart contracts in the light of contract law. *European Review of Private Law*, 26(6), 2018.

[70] Mingxiao Du, Qijun Chen, Lietong Liu, and Xiaofeng Ma. A blockchain-based random number generation algorithm and the application in blockchain games. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3498–3503, 2019. doi: 10.1109/SMC.2019.8914618.

[71] Aaron Edlin and Stefan Reichelstein. Holdups, standard breach remedies, and optimal investment, 1995.

[72] Sandro Etalle, Jeremy den Hartog, and Stephen Marsh. Trust and punishment. In *1st International Conference on Autonomic Computing and Communication Systems, Autonomics*, pages 5–1. ICST, 2007.

[73] S. Farshidi, S. Jansen, S. Espana, and J. Verkleij. Decision support for blockchain platform selection: Three industry case studies. *IEEE Transactions on Engineering Management*, 67(4):1109–1128, 2020. doi: 10.1109/TEM.2019.2956897.

[74] Christian Finke, Marie Alfken, and Matthias Schumann. Why do distributed ledger platforms fail? analyzing the challenges of distributed ledger technologies in supply chain processes. In *ACIS 2023 Proceedings*, 12 2023.

[75] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, apr 1985. ISSN 0004-5411. doi: 10.1145/3149.214121.

[76] N. Fotiou and G.C. Polyzos. Smart contracts for the internet of things: Opportunities and challenges. In *EuCNC 2018*, pages 256–260, 2018. doi: 10.1109/EuCNC.2018.8443212.

[77] Amit Ganeriwalla, Michael Casey, Prema Shrikrishna, Jan Philipp Bender, and Stefan Gstettner. Does your supply chain need a blockchain? Technical report, The Boston Consulting Group, 2018.

[78] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda, and V. Santamaria. To blockchain or not to blockchain: That is the question. *IT Professional*, 20(2): 62–74, 2018. doi: 10.1109/MITP.2018.021921652.

[79] R. Gelashvili, A. Spiegelman, Z. Xiang, G. Danezis, Z. Li, D. Malkhi, Y. Xia, and R. Zhou. Block-stm scaling blockchain execution by turning ordering curse to a performance blessing. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP*, pages 232–244, 2023. doi: 10.1145/3572848.3577524.

[80] M. Giancaspro. Is a 'smart contract' really a smart idea? insights from a legal perspective. *Computer Law and Security Review*, 33(6):825–835, 2017. doi: 10.1016/j.clsr.2017.05.007.

[81] Alex Gluchowski. Introducing zksync: the missing link to mass adoption of ethereum, 2019. URL https://blog.matter-labs.io/introducing-zk-sync-the-missing-link-to-mass-adoption-of-ethereum-14c9cea83f58.

[82] C. Gorenflo, S. Lee, L. Golab, and S. Keshav. Fastfabric: Scaling hyperledger fabric to 20 000 transactions per second. *International Journal of Network Management*, 30(5), 2020. doi: 10.1002/nem.2099.

[83] S.N.G. Gourisetti, M. Mylrea, and H. Patangia. Evaluation and demonstration of blockchain applicability framework. *IEEE Transactions on Engineering Management*, 67(4):1142–1156, 2020. doi: 10.1109/TEM.2019.2928280.

[84] G. Governatori, F. Idelberger, Z. Milosevic, R. Riveret, G. Sartor, and X. Xu. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law*, 26(4):377–409, 2018. doi: 10.1007/s10506-018-9223-3.

[85] Tobias Guggenberger, Johannes Sedlmeir, Gilbert Fridgen, and Andre Luckow. An in-depth investigation of the performance characteristics of hyperledger fabric. *Computers & Industrial Engineering*, page 108716, 2022. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2022.108716.

[86] Z. Guo, C. Hong, M. Yang, D. Zhou, L. Zhou, and L. Zhuang. Rex: Replication at the speed of multi-core. In *Proceedings of the 9th European Conference on Computer Systems, EuroSys 2014*, 2014. doi: 10.1145/2592798.2592800.

[87] H. Halaburda. Economic and business dimensions blockchain revolution without the blockchain? most of the suggested benefits of blockchain technologies do not come from elements unique to the blockchain. *Communications of the ACM*, 61(7):27–29, 2018. doi: 10.1145/3225619.

[88] Yuechen Hao. Research of the 51% attack based on blockchain. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*, pages 278–283, 2022. doi: 10.1109/CVIDLICCEA56201.2022.9824528.

[89] Abid Hassan, Md. Iftekhar Ali, Rifat Ahammed, Mohammad Monirujjaman Khan, Nawal Alsufyani, and Abdulmajeed Alsufyani. Secured insurance framework using blockchain and smart contract. *Scientific Programming*, 2021, 2021.

[90] V. Hassija, S. Zeadally, I. Jain, A. Tahiliani, V. Chamola, and S. Gupta. Framework for determining the suitability of blockchain: Criteria and issues to consider. *Transactions on Emerging Telecommunications Technologies*, 32 (10), 2021. doi: 10.1002/ett.4334.

[91] M. Havinga, M. Hoving, and V. Swagemakers. *Alibaba: A case study on building an international imperium on information and e-commerce*. Springer, 2016. doi: 10.1007/978-3-319-23012-2_2.

[92] Ming He, Haodi Wang, Yunchuan Sun, Rongfang Bie, Tian Lan, Qi Song, Xi Zeng, Matevzž Pustiseˇk, and Zhenyu Qiu. T2l: A traceable and trustable consortium blockchain for logistics. *Digital Communications and Networks*, 2022. ISSN 2352-8648. doi: https://doi.org/10.1016/j.dcan.2022.06.015.

[93] Tharaka Mawanane Hewa, Yining Hu, Madhusanka Liyanage, Salil S Kanhare, and Mika Ylianttila. Survey on blockchain-based smart contracts: Technical aspects and future research. *IEEE Access*, 9:87643–87662, 2021.

[94] M. Hribernik, K. Zero, S. Kummer, and D.M. Herold. City logistics: Towards a blockchain decision framework for collaborative parcel deliveries in micro-hubs. *Transportation Research Interdisciplinary Perspectives*, 8, 2020. ISSN 25901982. doi: 10.1016/j.trip.2020.100274.

[95] Bin Hu, Zongyang Zhang, Jianwei Liu, Yizhong Liu, Jiayuan Yin, Rongxing Lu, and Xiaodong Lin. A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns*, 2(2):100179, 2021. ISSN 2666-3899. doi: https://doi.org/10.1016/j.patter.2020.100179.

[96] Kai Hu, Jian Zhu, Yi Ding, Xiaomin Bai, and Jiehua Huang. Smart contract engineering. *Electronics*, 9(12):2042, 2020.

[97] Y.-C. Hu, T.-T. Lee, D. Chatzopoulos, and P. Hui. Analyzing smart contract interactions and contract level state consensus. *Concurrency and Computation: Practice and Experience*, 32(12), 2020. doi: 10.1002/cpe.5228.

[98] J.J. Hunhevicz and D.M. Hall. Do you need a blockchain in construction? use case categories and decision framework for dlt design options. *Advanced Engineering Informatics*, 45, 2020. doi: 10.1016/j.aei.2020.101094.

[99] Hyperledger. Hyperledger sawtooth blockchain security (part one), 2018. URL https://www.hyperledger.org/blog/2018/11/09/hyperledger-sawtooth-blockchain-security-part-one.

[100] Hyperledger. A blockchain platform for the enterprise, 2020. URL https://hyperledger-fabric.readthedocs.io/en/release-2.2/.

[101] Hyperledger Besu community. Besu enterprise ethereum client, 2021. URL https://besu.hyperledger.org/en/stable/.

[102] Hyperledger Caliper. Getting started, 2022. URL https://hyperledger.github.io/caliper/v0.5.0/getting-started/.

[103] Hyperledger Performance and Scale Working Group. Hyperledger blockchain performance metrics, 2019. URL https://www.hyperledger.org/resources/publications/blockchain-performance-metrics.

[104] Christopher J Ibbott and Robert M O'Keefe. Trust, planning and benefits in a global interorganizational system. *Information Systems Journal*, 14(2): 131–152, 2004.

[105] Intel Corporation. Introduction, 2017. URL https://sawtooth.hyperledger.org/docs/core/releases/1.2.6/introduction.html.

[106] A.U. Janssen and F.P. Patti. Demystifying smart contracts [demistificare gli smart contracts]. *Osservatorio del Diritto Civile e Commerciale*, 9(1):31–50, 2020. doi: 10.4478/98131.

[107] Ya-wen Jeng, Yung-chen Hsieh, and Ja-Ling Wu. Step-by-step guidelines for making smart contract smarter. In *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 25–32, 2019.

[108] C. Jin, S. Pang, X. Qi, Z. Zhang, and A. Zhou. A high performance concurrency protocol for smart contracts of permissioned blockchain. *IEEE Transactions on Knowledge and Data Engineering*, 34(11):5070–5083, 2022. doi: 10.1109/TKDE.2021.3059959.

[109] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, and M. Dahlin. All about eve: Execute-verify replication for multi-core servers. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012*, pages 237–250, 2012.

[110] Victor Youdom Kemmoe, William Stone, Jeehyeong Kim, Daeyoung Kim, and Junggab Son. Recent advances in smart contracts: A technical overview and state of the art. *IEEE Access*, 8:117782–117801, 2020.

[111] Abdullah Ayub Khan, Asif Ali Laghari, De-Sheng Liu, Aftab Ahmed Shaikh, Dan-An Ma, Chao-Yang Wang, and Asif Ali Wagan. Eps-ledger: Blockchain hyperledger sawtooth-enabled distributed power systems chain of operation and control node privacy and security. *Electronics*, 10(19), 2021. ISSN 2079-9292. doi: 10.3390/electronics10192395.

[112] T. Koens and E. Poll. What blockchain alternative do you need? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11025 LNCS:113–129, 2018. doi: 10.1007/978-3-030-00305-0_9.

[113] Bruce Kogut and Udo Zander. What firms do? coordination, identity, and learning. *Organization science*, 7(5):502–518, 1996.

[114] Jaturong Kongmanee, Phongphun Kijsanayothin, and Rattikorn Hewett. Securing smart contracts in blockchain. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, pages 69–76, 2019.

[115] Ramakrishna Kotla and Mike Dahlin. High throughput byzantine fault tolerance. In *Proceedings of the International Conference on Dependable Systems and Networks*, page 575 – 584, 2004. doi: 10.1109/dsn.2004.1311928.

[116] Devinder Kumar, Rajesh Kr Singh, Ruchi Mishra, and Tugrul U. Daim. Roadmap for integrating blockchain with internet of things (iot) for sustainable and secured operations in logistics and supply chains: Decision making framework with case illustration. *Technological Forecasting and Social Change*, 196, 2023. doi: 10.1016/j.techfore.2023.122837.

[117] Jae Kwon and Ethan Buchman. A network of distributed ledgers, 2019. URL https://v1.cosmos.network/resources/whitepaper.

[118] M. Kwon and H. Yu. Performance improvement of ordering and endorsement phase in hyperledger fabric. In *2019 6th International Conference on Internet of Things: Systems, Management and Security, IOTSMS 2019*, pages 428–432, 2019. doi: 10.1109/IOTSMS48152.2019.8939202.

[119] O. Labazova. Towards a framework for evaluation of blockchain implementations. In *40th International Conference on Information Systems, ICIS 2019*, 2019.

[120] O. Labazova, T. Dehling, and A. Sunyaev. From hype to reality: A taxonomy of blockchain applications. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, volume 2019-January, pages 4555–4564, 2019.

[121] Pascal Lafourcade and Marius Lombard-Platet. About blockchain interoperability. *Information Processing Letters*, 161:105976, 05 2020. doi: 10.1016/j.ipl.2020.105976.

[122] A. Lakshman and P. Malik. Cassandra - a decentralized structured storage system. In *Operating Systems Review (ACM)*, volume 44, pages 35–40, 2010. doi: 10.1145/1773912.1773922.

[123] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[124] Jan Lánskỳ. Bitcoin system. *Acta Informatica Pragensia*, 6(1):20–31, 2017.

[125] Dan Larimer. Eos. io technical white paper v2. Technical report, EOS.IO, 2018.

[126] Kristian Lauslahti, Juri Mattila, and Timo Seppälä. Smart contracts – how will blockchain technology affect contractual practices? ETLA Report 68, Helsinki, 2017. URL http://hdl.handle.net/10419/201350.

[127] S. Li, B. Ragu-Nathan, T.S. Ragu-Nathan, and S. Subba Rao. The impact of supply chain management practices on competitive advantage and organizational performance. *Omega*, 34(2):107–124, 2006. doi: 10.1016/j.omega.2004.08.002.

[128] I.-C. Lin and T.-C. Liao. A survey of blockchain security issues and challenges. *International Journal of Network Security*, 19(5):653–659, 2017. doi: 10.6633/IJNS.201709.19(5).01.

[129] Linux Foundation. Case studies, 2020. URL https://www.hyperledger.org/learn/case-studies.

[130] Linux Foundation. About hyperledger, 2020. URL https://www.hyperledger.org/about.

[131] Yang Liu and Jincheng Huang. Legal creation of smart contracts and the legal effects. In *Journal of Physics: Conference Series*, volume 1345, page 042033, 2019.

[132] S.K. Lo, X. Xu, Y.K. Chiam, and Q. Lu. Evaluating suitability of applying blockchain. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, volume 2017-November, pages 158–161, 2017. doi: 10.1109/ICECCS.2017.26.

[133] F. Lumineau, W. Wang, and O. Schilke. Blockchain governance-a new way of organizing collaborations? *Organization Science*, 32(2):500–521, 2021. doi: 10.1287/orsc.2020.1379.

[134] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269, 2016.

[135] J. Ma, Y. Jo, and C. Park. Peerbft: Making hyperledger fabric's ordering service withstand byzantine faults. *IEEE Access*, 8:217255–217267, 2020. doi: 10.1109/ACCESS.2020.3040443.

[136] Daniel Macrinici, Cristian Cartofeanu, and Shang Gao. Smart contract applications within blockchain technology: A systematic mapping study. *Telematics and Informatics*, 35(8):2337–2354, 2018.

[137] P.J. Marandi and F. Pedone. Optimistic parallel state-machine replication. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, volume 2014-January, pages 57–66, 2014. doi: 10.1109/SRDS.2014.25.

[138] P.J. Marandi, C.E. Bezerra, and F. Pedone. Rethinking state-machine replication for parallelism. In *Proceedings - International Conference on Distributed Computing Systems*, pages 368–377, 2014. doi: 10.1109/ICDCS.2014.45.

[139] Bill Marino and Ari Juels. Setting standards for altering and undoing smart contracts. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 151–166, 2016.

[140] Jack Martin. Bitcoin gold blockchain hit by 51% attack leading to $70k double spend, 2020. URL https://cointelegraph.com/news/bitcoin-gold-blockchain-hit-by-51-attack-leading-to-70k-double-spend.

[141] Marco Mazzoni, Antonio Corradi, and Vincenzo Di Nicola. Performance evaluation of permissioned blockchains for financial applications: The consensys quorum case study. *Blockchain: Research and Applications*, 3(1):100026, 2022. doi: https://doi.org/10.1016/j.bcra.2021.100026.

[142] Scott A McKinney, Rachel Landy, and Rachel Wilka. Smart contracts, blockchain, and the next frontier of transactional law. *Wash. JL Tech. & Arts*, 13:313, 2017.

[143] Daniel Mera. Quorum blockchain stress evaluation in different environments, 2019. URL https://academicworks.cuny.edu/cgi/viewcontent.cgi?article=1120&context=jj_etds.

[144] E. Mik. Smart contracts: terminology, technical limitations and real world complexity. *Law, Innovation and Technology*, 9(2):269–300, 2017. doi: 10.1080/17579961.2017.1378468.

[145] A.A. Monrat, O. Schelen, and K. Andersson. Performance evaluation of permissioned blockchain platforms. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering, CSDE 2020*, pages 1–8, 2020. doi: 10.1109/CSDE50874.2020.9411380.

[146] Hans Moog. A new "consensus": The tangle multiverse [part 1], 2019. URL https://husqy.medium.com/a-new-consensus-the-tangle-multiverse-part-1-da4cb2a69772.

[147] Shashank Motepalli and Hans-Arno Jacobsen. Decentralizing permissioned blockchain with delay towers. *arXiv preprint arXiv:2203.09714*, 2022.

[148] M. Nadini, L. Alessandretti, F. Di Giacinto, M. Martino, L.M. Aiello, and A. Baronchelli. Mapping the nft revolution: market trends, trade networks, and visual features. *Scientific Reports*, 11(1), 2021. doi: 10.1038/s41598-021-00053-8.

[149] S. Naef, S.M. Wagner, and C. Saur. Blockchain and network governance: learning from applications in the supply chain sector. *Production Planning and Control*, 2022. doi: 10.1080/09537287.2022.2044072.

[150] T. Nakaike, Q. Zhang, Y. Ueda, T. Inagaki, and M. Ohara. Hyperledger fabric performance characterization and optimization using goleveldb benchmark. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020*, pages 1–9, 2020. doi: 10.1109/ICBC48266.2020.9169454.

[151] Q. Nasir, I.A. Qasse, M. Abu Talib, and A.B. Nassif. Performance analysis of hyperledger fabric platforms. *Security and Communication Networks*, 2018, 2018. doi: 10.1155/2018/3976093.

[152] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, and P. Jayachandran. Blockchain meets database: Design and implementation of a blockchain relational database. In *Proceedings of the VLDB Endowment*, volume 12, pages 1539–1552, 2018. doi: 10.14778/3342263.3342632.

[153] Alex Norta. Designing a smart-contract application layer for transacting decentralized autonomous organizations. In *International Conference on Advances in Computing and Data Sciences*, pages 595–604, 2016.

[154] O.C. Novac, D.E. Madar, C.M. Novac, G. Bujdoso, M. Oproescu, and T. Gal. Comparative study of some applications made in the angular and vue.js frameworks. In *2021 16th International Conference on Engineering of Modern Electric Systems, EMES 2021 - Proceedings*, 2021. doi: 10.1109/EMES52337.2021.9484150.

[155] Kelly Olson, Mic Bowman, James Mitchell, Shawn Amundson, Dan Middleton, and Cian Montgomery. Sawtooth: An introduction, 2018. URL https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf.

[156] Eric Olszewski. Why blockchain matters to enterprise (hint: It's not because of decentralization), 2019. URL https://medium.com/@eolszewski/why-blockchain-matters-to-enterprise-hint-its-not-because-of-decentralization-8c38674f43c6.

[157] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference, USENIX ATC 2014*, pages 305–319, 2014.

[158] Michael Paik, Jerónimo Irazábal, Dennis Zimmer, Michele Meloni, and Valentin Padurean. immudb: A lightweight, performant immutable database, 2020. URL https://arxiv.org/pdf/2207.06870.pdf.

[159] D. Pan, J.L. Zhao, S. Fan, and Z. Zhang. Dividend or no dividend in delegated blockchain governance: A game theoretic analysis. *Journal of Systems Science and Systems Engineering*, 30(3):288–306, 2021. doi: 10.1007/s11518-021-5487-3.

[160] R. Pan, M. Duan, C. Liu, K. Li, G. Xiao, and K. Li. An algorithm and architecture co-design for accelerating smart contracts in blockchain. In *Proceedings - International Symposium on Computer Architecture*, pages 446–458, 2023. doi: 10.1145/3579371.3589067.

[161] M.E. Peck. Blockchain world - do you need a blockchain? this chart will tell you if the technology can solve your problem. *IEEE Spectrum*, 54(10):38–60, 2017. doi: 10.1109/MSPEC.2017.8048838.

[162] A.B. Pedersen, M. Risius, and R. Beck. A ten-step decision path to determine when to use blockchain technologies. *MIS Quarterly Executive*, 18(2):99–115, 2019. doi: 10.17705/2msqe.00010.

[163] Rowan van Pelt, Slinger Jansen, Djuri Baars, and Sietse Overbeek. Defining blockchain governance: a framework for analysis and comparison. *Information Systems Management*, 38(1):21–41, 2021.

[164] Z. Peng, Y. Zhang, Q. Xu, H. Liu, Y. Gao, X. Li, and G. Yu. Neuchain: A fast permissioned blockchain system with deterministic ordering. *Proceedings of the VLDB Endowment*, 15(11):2585–2598, 2022. doi: 10.14778/3551793.3551816.

[165] G. Perboli, S. Musso, and M. Rosano. Blockchain in logistics and supply chain: A lean approach for designing real-world use cases. *IEEE Access*, 6: 62018–62028, 2018. doi: 10.1109/ACCESS.2018.2875782.

[166] Guido Perboli, Vittorio Capocasale, and Danilo Gotta. Blockchain-based transaction management in smart logistics: A sawtooth framework. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1713–1718, 2020. doi: 10.1109/COMPSAC48688.2020.000-8.

[167] Permissioned Distributed Ledger ETSI Industry Specification Group. Etsi gs pdl 011 v1.1.1. Technical report, ETSI, 2021.

[168] J. Polge, J. Robert, and Y. Le Traon. Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, 7(2):229–233, 2021. doi: 10.1016/j.icte.2020.09.002.

[169] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong. Performance analysis of private blockchain platforms in varying workloads. In *2017 26th International Conference on Computer Communications and Networks, ICCCN 2017*, pages 1–6, 2017. doi: 10.1109/ICCCN.2017.8038517.

[170] M.E. Porter. The five competitive forces that shape strategy. *Harvard Business Review*, 86(1):79–93+137, 2008.

[171] D. Puthal, S.P. Mohanty, E. Kougianos, and G. Das. When do we need the blockchain? *IEEE Consumer Electronics Magazine*, 10(2):53–56, 2021. doi: 10.1109/MCE.2020.3015606.

[172] M. Rasolroveicy and M. Fokaefs. Performance evaluation of distributed ledger technologies for iot data registry: A comparative study. In *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability, WS4 2020*, pages 137–144, 2020. doi: 10.1109/WorldS450073.2020.9210358.

[173] R. Ratasuk, N. Mangalvedhe, Y. Zhang, M. Robert, and J.-P. Koskinen. Overview of narrowband iot in lte rel-13. In *IEEE Conference on Standards for Communications and Networking*, 2016. doi: 10.1109/CSCN.2016.7785170.

[174] Michel Rauchs, Apolline Blandin, Keith Bear, and Stephen B McKeon. 2nd global enterprise blockchain benchmarking study, 2019. URL https://cdn.crowdfundinsider.com/wp-content/uploads/2019/09/2019-ccaf-second-global-enterprise-blockchain-report.pdf.

[175] P. Ritala, A. Golnam, and A. Wegmann. Coopetition-based business models: The case of amazon.com. *Industrial Marketing Management*, 43(2):236–249, 2014. doi: 10.1016/j.indmarman.2013.11.005.

[176] Sara Rouhani and Ralph Deters. Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access*, 7:50759–50779, 2019.

[177] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, and B.C. Ooi. A transactional perspective on execute-order-validate blockchains. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 543–557, 2020. doi: 10.1145/3318464.3389693.

[178] Magne Saetran, Jungwon Seo, and Sooyong Park. Leverage sidechains to reduce the workload of smart contracts through parallelization. *Journal of Computing Science and Engineering*, 15(3):125–133, 2021.

[179] D. Saingre, T. Ledoux, and J.-M. Menaud. Bctmark: A framework for benchmarking blockchain technologies. In *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, volume 2020-November, pages 1–8, 2020. doi: 10.1109/AICCSA50499.2020.9316536.

[180] Pablo Sanz Bayón. Key legal issues surrounding smart contract applications. *KLRI Journal of Law and Legislation*, 9(1):63–91, 2019.

[181] Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. Smart contract: Attacks and protections. *IEEE Access*, 8:24416–24427, 2020.

[182] Paul Schaaf, Filip Rezabek, and Holger Kinkelin. Analysis of proof of stake flavors with regards to the scalability trilemma. *Network*, 63, 2021.

[183] Fabian Schär. Decentralized finance: On blockchain-and smart contract-based financial markets. *FRB of St. Louis Review*, 2021.

[184] B. Schneider and W. Azan. Perceptions and misconceptions of blockchain: The potential of applying threshold concept theory. In *2022 IEEE 6th International Conference on Logistics Operations Management, GOL 2022*, 2022. doi: 10.1109/GOL53975.2022.9820452.

[185] F.B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990. doi: 10.1145/98163.98167.

[186] F. Schuhknecht, A. Sharma, J. Dittrich, and D. Agrawal. chainifydb: How to get rid of your blockchain and use your dbms instead. In *11th Annual Conference on Innovative Data Systems Research, CIDR 2021*, 2021.

[187] B.A. Scriber. A framework for determining blockchain applicability. *IEEE Software*, 35(4):70–77, 2018. doi: 10.1109/MS.2018.2801552.

[188] Johannes Sedlmeir, Philipp Ross, André Luckow, Jannik Lockl, Daniel Miehle, and Gilbert Fridgen. The dlps: A new framework for benchmarking blockchains. In *HICSS*, pages 1–10, 2021.

[189] I. Sergey, V. Nagaraj, J. Johannsen, A. Kumar, A. Trunov, and K.C.G. Hao. Safer smart contract programming with scilla. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA), 2019. doi: 10.1145/3360611.

[190] J. Shah and D. Sharma. Performance benchmarking frameworks for distributed ledger technologies. In *Proceedings of CONECCT 2021: 7th IEEE International Conference on Electronics, Computing and Communication Technologies*, pages 1–5, 2021. doi: 10.1109/CONECCT52877.2021.9622659.

[191] S. Shalaby, A.A. Abdellatif, A. Al-Ali, A. Mohamed, A. Erbad, and M. Guizani. Performance evaluation of hyperledger fabric. In *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies, ICIoT 2020*, pages 608–613, 2020. doi: 10.1109/ICIoT48696.2020.9089614.

[192] Guangzhi Shang, Noyan Ilk, and Shaokun Fan. Need for speed, but how much does it cost? unpacking the fee-speed relationship in bitcoin transactions. *Journal of Operations Management*, 69(1):102–126, 2023.

[193] G. Shapiro, C. Natoli, and V. Gramoli. The performance of byzantine fault tolerant blockchains. In *2020 IEEE 19th International Symposium on Network Computing and Applications, NCA 2020*, pages 1–8, 2020. doi: 10.1109/NCA51143.2020.9306742.

[194] A. Sharma, D. Agrawal, F.M. Schuhknecht, and J. Dittrich. Blurring the lines between blockchains and database systems: The case of hyperledger fabric. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 105–122, 2019. doi: 10.1145/3299869.3319883.

[195] Z. Shi, H. Zhou, Y. Hu, S. Jayachander, C. De Laat, and Z. Zhao. Operating permissioned blockchain in clouds: A performance study of hyperledger sawtooth. In *Proceedings - 2019 18th International Symposium on Parallel and Distributed Computing, ISPDC 2019*, pages 50–57, 2019. doi: 10.1109/ISPDC.2019.00010.

[196] J. Sousa, A. Bessani, and M. Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018*, pages 51–58, 2018. doi: 10.1109/DSN.2018.00018.

[197] Puli Sreehari, M Nandakishore, Goutham Krishna, Joshin Jacob, and VS Shibu. Smart will converting the legal testament into a smart contract. In *2017 International Conference on Networks & Advances in Computational Technologies (NetACT)*, pages 203–207, 2017.

[198] H.S. Sternberg, E. Hofmann, and D. Roeck. The struggle is real: Insights from a supply chain blockchain case. *Journal of Business Logistics*, 42(1): 71–87, 2021. doi: 10.1111/jbl.12240.

[199] H. Sukhwani, N. Wang, K.S. Trivedi, and A. Rindos. Performance modeling of hyperledger fabric (permissioned blockchain network). In *NCA 2018 - 2018 IEEE 17th International Symposium on Network Computing and Applications*, pages 1–8, 2018. doi: 10.1109/NCA.2018.8548070.

[200] T. Sund, C. Lööf, S. Nadjm-Tehrani, and M. Asplund. Blockchain-based event processing in supply chains—a case study at ikea. *Robotics and Computer-Integrated Manufacturing*, 65, 2020. doi: 10.1016/j.rcim.2020.101971.

[201] N. Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997. doi: 10.5210/fm.v2i9.548.

[202] Péter Szilágyi. Eip-225: Clique proof-of-authority consensus protocol, 2017. URL https://eips.ethereum.org/EIPS/eip-225.

[203] TechCrunch. Market analysis: Stellantis circular economy, 2022. URL https://techcrunch.com/2022/10/11/stellantis-says-circular-economy-business-unit-will-rake-in-revenue-of-e2b-by-2030/.

[204] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. Performance benchmarking and optimizing hyperledger fabric blockchain platform. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 264–276, 2018.

[205] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li. A survey of smart contract formal specification and verification. *ACM Computing Surveys*, 54(7), 2022. doi: 10.1145/3464421.

[206] Jesus Leal Trujillo, Steve Fromhart, and Val Srinivas. Evolution of blockchain technology, 2017. URL https://www2.deloitte.com/us/en/insights/industry/financial-services/evolution-of-blockchain-github-platform.html.

[207] A. Valtchanov, L. Helbling, B. Mekiker, and M.P. Wittie. Parallel block execution in socc blockchains through optimistic concurrency control. In *2021 IEEE Globecom Workshops, GC Wkshps 2021 - Proceedings*, 2021. doi: 10.1109/GCWkshps52748.2021.9682147.

[208] Nidish Vashistha, Muhammad Monir Hossain, Md Rakib Shahriar, Farimah Farahmandi, Fahim Rahman, and Mark Tehranipoor. echain: A blockchain-enabled ecosystem for electronic device authenticity verification. *IEEE Transactions on Consumer Electronics*, 2021.

[209] Zach Voell. Ethereum classic hit by third 51% attack in a month, 2021. URL https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month/.

[210] C. Wang and X. Chu. Performance characterization and bottleneck analysis of hyperledger fabric. In *Proceedings - International Conference on Distributed Computing Systems*, volume 2020-November, pages 1281–1286, 2020. doi: 10.1109/ICDCS47774.2020.00165.

[211] S. Wang, M. Yang, Y. Zhang, Y. Luo, T. Ge, X. Fu, and W. Zhao. On private data collection of hyperledger fabric. In *Proceedings - International Conference on Distributed Computing Systems*, volume 2021-July, pages 819–829, 2021. doi: 10.1109/ICDCS51616.2021.00083.

[212] Zhiqiang Wang, Fei Ye, and Kim Hua Tan. Effects of managerial ties and trust on supply chain information sharing and supplier opportunism. *International journal of production Research*, 52(23):7046–7061, 2014.

[213] X. Wei, S. Junyi, Z. Qi, and C. Fu. Xuperchain: A blockchain system that supports smart contracts parallelization. In *Proceedings - 2020 IEEE International Conference on Smart Internet of Things, SmartIoT 2020*, pages 309–313, 2020. doi: 10.1109/SmartIoT49966.2020.00055.

[214] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger, 2021. URL https://ethereum.github.io/yellowpaper/paper.pdf.

[215] K. Wust and A. Gervais. Do you need a blockchain? In *Proceedings - 2018 Crypto Valley Conference on Blockchain Technology, CVCBT 2018*, pages 45–54, 2018. doi: 10.1109/CVCBT.2018.00011.

[216] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

[217] C. Xu, C. Zhang, J. Xu, and J. 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.

[218] Jeff Hoi Yan Yeung, Willem Selen, Min Zhang, and Baofeng Huo. The effects of trust and coercive power on supplier integration. *International journal of production Economics*, 120(1):66–78, 2009.

[219] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W.J. Knottenbelt. Sok: Communication across distributed ledgers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12675 LNCS:3–36, 2021. doi: 10.1007/978-3-662-64331-0_1.

[220] Florian Zemler. Concepts for gdpr-compliant processing of personal data on blockchain: a literature review. *Anwendungen und Konzepte der Wirtschaftsinformatik*, 10:12–12, 2019.

[221] Haowen Zhang, Jing Li, He Zhao, Tong Zhou, Nianzu Sheng, and Hengyu Pan. Blockpilot: A proposer-validator parallel execution framework for blockchain. In *ACM International Conference Proceeding Series*, page 193 – 202, 2023. doi: 10.1145/3605573.3605621.

[222] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018. doi: 10.1504/IJWGS.2018.095647.

[223] H. Zhong, Y. Sang, Y. Zhang, and Z. Xi. Secure multi-party computation on blockchain: An overview. *Communications in Computer and Information Science*, 1163:452–460, 2020. doi: 10.1007/978-981-15-2767-8_40.

[224] W. Zou, D. Lo, P.S. Kochhar, X.-B.D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 47(10):2084–2106, 2021. doi: 10.1109/TSE.2019.2942301.