Technische Universität München

TUM School of Computation, Information and Technology

Holistic Approaches to Performance Optimization in Decentralized Systems: A Study of Hyperledger Fabric

Jeeta Ann Chacko

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technische Universität München zur Erlangung des akademischen Grades einer

Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Michael Gerndt

Prüfende der Dissertation:

1. Prof. Dr. Hans-Arno Jacobsen
2. Prof. Dr. Florian Matthes
3. Prof. Dr. Bettina Kemme

Die Dissertation wurde am 24.06.2024 bei der Technische Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 29.10.2024 angenommen.

Abstract

Despite offering valuable features such as decentralized trust, immutability, and traceability, the adoption of decentralized solutions remains limited compared to conventional transaction processing systems, primarily due to significant performance disparities. This dissertation explores holistic approaches to optimize the performance of decentralized systems, utilizing Hyperledger Fabric, a prominent blockchain system, as a case study.

In the initial phase of our research, we investigate transaction failures in Hyperledger Fabric. We formally define various failure types and conduct extensive experiments to study the impact of diverse system configurations on transaction outcomes. Based on our analysis, we provide actionable guidelines to enhance the system's performance and reliability.

Building on this foundation, our second study introduces a novel, multi-level optimization strategy encompassing system, data, and user dimensions. By analyzing the blockchain ledger, nine cogent optimization recommendations are derived and encapsulated within our automated tool, BlockOptR. The implementation of these recommendations significantly improves transaction success rates and reduces latency, underscoring the efficacy of our comprehensive optimization approach.

In the third segment of our research, we investigate the concept of self-driving blockchains. A framework is proposed to enable Hyperledger Fabric to autonomously predict and adapt to workload variations, thereby optimizing its performance in real time. Experiments conducted across different blockchain layers validate the potential of this autonomous approach, demonstrating substantial enhancements in throughput and latency.

Additionally, this research includes the development of a visualization tool for transactions within a Fabric network, aiding users in understanding the characteristics of their workloads. We also conducted a comprehensive study on benchmarking different blockchains to provide a broader perspective on various decentralized systems beyond Fabric. Collectively, this dissertation offers a robust foundation for advancing the performance of decentralized systems.

Zusammenfassung

Obwohl sie wertvolle Funktionen wie dezentrales Vertrauen, Unveränderlichkeit und Rückverfolgbarkeit bieten, ist die Akzeptanz dezentraler Lösungen im Vergleich zu herkömmlichen Transaktionsverarbeitungssystemen nach wie vor begrenzt, was vor allem auf erhebliche Leistungsunterschiede zurückzuführen ist. In dieser Dissertation werden ganzheitliche Ansätze zur Optimierung der Leistung dezentraler Systeme untersucht, wobei Hyperledger Fabric, ein bekanntes Blockchain-System, als Fallstudie verwendet wird.

In der ersten Phase unserer Forschung untersuchen wir Transaktionsausfälle in Hyperledger Fabric. Wir definieren formell verschiedene Fehlertypen und führen umfangreiche Experimente durch, um die Auswirkungen verschiedener Systemkonfigurationen auf die Transaktionsergebnisse zu untersuchen. Auf der Grundlage unserer Analyse liefern wir umsetzbare Richtlinien zur Verbesserung der Leistung und Zuverlässigkeit des Systems. Aufbauend auf dieser Grundlage führt unsere zweite Studie eine neuartige, mehrstufige Optimierungsstrategie ein, die System-, Daten- und Nutzerdimensionen umfasst. Durch die Analyse des Blockchain-Ledgers werden neun schlüssige Optimierungsempfehlungen abgeleitet und in unserem automatisierten Tool BlockOptR gekapselt. Die Umsetzung dieser Empfehlungen verbessert die Erfolgsquote von Transaktionen erheblich und verringert die Latenzzeit, was die Wirksamkeit unseres umfassenden Optimierungsansatzes unterstreicht.

Im dritten Teil unserer Forschung untersuchen wir das Konzept der selbststeuernden Blockchains. Es wird ein Rahmenwerk vorgeschlagen, das Hyperledger Fabric in die Lage versetzt, selbstständig Arbeitslastschwankungen vorherzusagen und sich an diese anzupassen, um so die Leistung in Echtzeit zu optimieren. Experimente, die über verschiedene Blockchain-Schichten hinweg durchgeführt wurden, bestätigen das Potenzial dieses autonomen Ansatzes und zeigen erhebliche Verbesserungen bei Durchsatz und Latenzzeit. Darüber hinaus umfasst diese Forschung die Entwicklung eines Visualisierungstools für Transaktionen innerhalb eines Fabric-Netzwerks, das den Benutzern hilft, die Eigenschaften ihrer Arbeitslasten zu verstehen. Wir haben auch eine umfassende Studie zum Benchmarking verschiedener Blockchains durchgeführt, um eine breitere Perspektive auf verschiedene dezentralisierte Systeme jenseits von Fabric zu bieten. Diese Dissertation bietet eine solide Grundlage für die Weiterentwicklung der Leistung dezentraler Systeme.

Acknowledgments

I want to express my sincere gratitude to the supportive community of family, friends, colleagues, collaborators, students, and acquaintances who have collectively contributed, in both small and large ways, to making this doctoral journey a meaningful experience.

My heartfelt appreciation goes to Prof. Hans-Arno Jacobsen for believing in my potential and accepting my application for this doctoral program. His constant guidance and motivation over the years have been invaluable. I also thank Prof. Ruben Mayer for the numerous discussions that significantly helped me formulate well-defined research goals, and for his valuable and actionable feedback on all my work.

I extend my thanks to Prof. Dr. Florian Matthes and Prof. Dr. Bettina Kemme for agreeing to be the examiners for my thesis. Special thanks to Prof. Michael Gerndt for chairing the thesis committee and for his guidance during my Masters, which sparked my interest in pursuing a doctorate. Additionally, I am grateful to Prof. Alan Fekete for the valuable collaboration and to Prof. Viktor Leis for his support towards the end of my doctorate.

I am deeply grateful to all my colleagues who created a friendly and supportive atmosphere. Special thanks to Christoph, Alex, and Herbert for maintaining our infrastructure, without which none of my experiments would have been possible. Thanks to René, Jana, Nikolai, Jawad, and Irene for all their help and fun discussions over the years. I am extremely thankful to Pezhman for opening the doorway to this doctoral position and for all our personal and academic discussions. I also extend my thanks to all the students I had the opportunity to teach and supervise, whose contributions, directly or indirectly, enriched my research.

My infinite gratitude goes to my boyfriend, Jophin, whose persistent encouragement and optimism gave me the strength to overcome all the barriers in both my academic and personal life. Thank you for being with me every step of the way. My heartfelt thanks to my sister (Chechi) for providing me with a wonderfully exciting childhood and for being a constant source of interesting discussions in my adulthood. I am deeply thankful to my parents (Mummy and Daddy) for all their sacrifices and for always believing in me. Everything I am and where I stand today, I owe it all to them.

Contents

1

Introduction

The advent of Bitcoin in 2009 marked the dawn of a new era in peer-to-peer systems, rekindling interest in the concepts of decentralized consensus, cryptographic proofs, and immutable ledgers [1]. The domain of blockchains underwent further revolution with the launch of Ethereum in 2015, which introduced the idea of smart contracts that could support complex transactions [2]. However, private enterprises were initially apprehensive about adopting this new technology due to the inherent open-access nature of blockchains. Moreover, blockchains and conventional transaction processing systems exhibit significant performance differences. Bitcoin can only handle around seven transactions per second (TPS), whereas Visa can handle transactions in the order of 24,000 TPS [3, 4].

A novel class of decentralized systems, commonly known as permissioned blockchains, has emerged as a potential solution to address the performance limitations and privacy concerns associated with traditional blockchain systems [5]. These systems enable access only to authorized users and are designed to provide superior performance. Permissioned blockchains, such as Hyperledger Fabric (also known as Fabric), Quorum, Corda, and Multichain, operate in a partially decentralized environment and offer faster transaction processing compared to Bitcoin or Ethereum [6, 7, 8, 9]. They provide immutable traceability with greater scalability and security, albeit at the cost of complete decentralization. Despite these advancements, permissioned blockchains are still unable

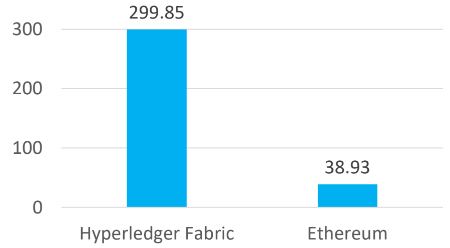


Figure 1.1.1: Throughput in TPS for Fabric vs. Ethereum. Data from [10]

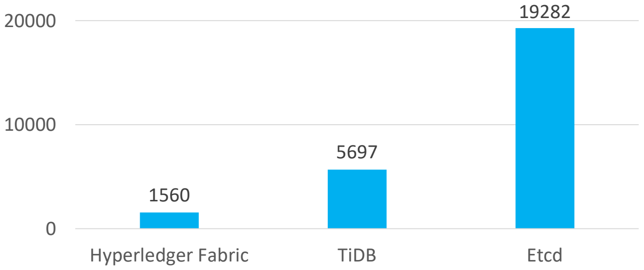


Figure 1.1.2: Throughput in TPS for Fabric vs. distributed database systems. Data from [11]

to match the performance of conventional transaction processing systems.

This work aims to address the limitations of permissioned blockchains and to derive performance optimization strategies that can enhance the throughput and latency of these systems. Our research employs experimental analysis to identify and formalize performance bottlenecks. Moreover, this work introduces a multi-level recommendation system and a self-driving system, both of which significantly improve the performance.

1.1 Motivation

The practical applications of blockchain technology in various domains, such as supply chain management, decentralized finance, and asset management, have been well-established [12, 13]. The Ethereum network boasts over 4,000 decentralized applications, and more than 200 systems have been developed using Hyperledger technologies [14, 15]. However, the adoption and acceptance of blockchain solutions pale in comparison to conventional transaction processing systems. This reluctance to embrace blockchain technology can be attributed to several factors, including high cost, complex design, low throughput, high latency, and limited scalability [16, 17, 18].

Permissioned blockchains, when compared to traditional public blockchains, offer superior performance. For instance, Fabric, one of the most popular blockchain platforms, provides substantially higher performance than Ethereum under similar network conditions (Figure 1.1.1). Nevertheless, when compared to distributed database systems such as

TiDB and Etcd, Fabric's performance remains significantly restricted (Figure 1.1.2). The salient and desirable characteristics of blockchains, like their immutability, traceability and decentralized trust, are enabled at the cost of this tradeoff with performance.

Blockchains are intricate systems that operate on a diverse set of concepts, such as smart contracts, cryptographic hash, immutable timestamped ledgers, decentralized consensus protocols, and access control rules [19, 20, 21, 22, 23]. Due to the presence of these various elements, identifying and comprehending the root causes of performance bottlenecks in such systems can be a challenging task. For instance, Fabric exhibits various types of transaction failures, and each of these failures is caused by bottlenecks at different levels of the blockchain stack [24]. These bottlenecks can be attributed to factors such as slow processing of transactions, network congestion, hardware limitations, and inadequate resource allocation. Therefore, identifying and understanding the specific limitations of blockchains and their causes is a significant research problem.

There are various ways to enhance the performance of decentralized systems. One such approach is the use of transaction conflict management strategies such as transaction reordering [25, 26]. This can be incorporated either at the data level in the smart contract or at the application level in the business process model in order to increase the success rate of transactions. Another approach is a system level strategy that involves dynamic client and peer resource management [27]. This can improve performance by allocating resources based on the incoming workload. Hence, identifying such effective optimization strategies at various levels of the blockchain stack is another challenging yet necessary task.

Optimization strategies for blockchain networks are highly dependent on the specific characteristics of the workloads they manage. Consequently, thoroughly understanding these workload characteristics is essential for identifying the most suitable optimization strategies [28]. Further, since workloads adapt over time, these strategies necessitate dynamic implementation and continuous human intervention, which results in increased costs [29, 30, 31]. Hence, there is a critical need for autonomous optimization strategies within blockchain environments. By continuously monitoring workload changes and dynamically applying optimization techniques, the performance of blockchain networks can be significantly enhanced.

Moreover, consistent and unbiased benchmarking is paramount for accurately assessing and comparing the efficacy of different blockchain systems and their corresponding optimization strategies. Without standardized and impartial evaluation metrics, it becomes challenging to gauge the true performance improvements offered by various optimization techniques.

1.2 Problem Statement

In order to improve the efficiency of decentralized systems, we identified three important research problems that need to be addressed. The first is studying performance limitations, which involves understanding the factors that are currently limiting the performance of blockchains. The second problem is deriving holistic optimization strategies that encompass the entire blockchain stack, from the underlying infrastructure to the application layer. Finally, there is a need to develop autonomous blockchain systems that can automatically adjust their performance in response to changing usage patterns without requiring manual intervention. By addressing these research problems, we can create more efficient and effective decentralized systems that can support a wide range of applications and use cases.

1.2.1 Identifying Performance Limitations

The use of experimental evaluation and benchmarking is crucial for understanding the performance capabilities of a system. However, when benchmarking, one must consider various aspects, such as parameter tuning, workload generation, and performance metrics. Given the complexity of the blockchain stack, it is essential to study the impact of parameter tuning on the system, data, and application levels of the blockchain. For instance, the block size, which refers to the number of transactions in a block, is a prominent system-level parameter, while the language of the smart contract is a critical data-level parameter, and both significantly influence performance [32]. Moreover, it is essential to use a comprehensive set of workloads for benchmarking to identify all limitations. For example, in Fabric, transaction failures due to dependency conflicts occur frequently, requiring a skewed workload to evaluate them effectively [25].

Additionally, the performance of blockchains can be measured using various metrics such as throughput, success rate, time to finality, scalability, and fault tolerance, each of which could have a different definition based on the blockchain implementation. For instance, successful transaction throughput is often defined as the number of transactions committed on the blockchain ledger per second. However for Fabric, committed transaction throughput and successful transaction throughput are two distinct metrics because even failed transactions are committed on the ledger [24]. In summary, benchmarking a blockchain system is a challenging task that requires a thorough understanding of its underlying implementation.

1.2.2 Deriving Performance Optimization Strategies

Current optimization attempts have primarily been confined to system and data level enhancements without a comprehensive strategy that addresses the multifaceted nature of blockchain ecosystems [33, 34]. This narrow focus overlooks critical areas such as governance structures and business process model efficiency that could significantly improve the overall functionality and adoption of blockchain systems.

To optimize blockchain operations, we need to adopt a broader perspective that aims to identify and implement improvements across multiple dimensions of the blockchain ecosystem. This approach should consider the entire blockchain stack to uncover optimization opportunities that traditional methods may miss. In other words, a holistic strategy is required for developing more robust, efficient, and scalable blockchain solutions.

1.2.3 Autonomous Dynamic Performance Optimization

Performance optimization strategies are important for blockchain systems, but their effectiveness is highly dependent on the workload, which often varies over time [35]. Additionally, blockchain systems must accommodate a constantly growing distributed ledger, leading to significant performance variations over time, even under a uniform

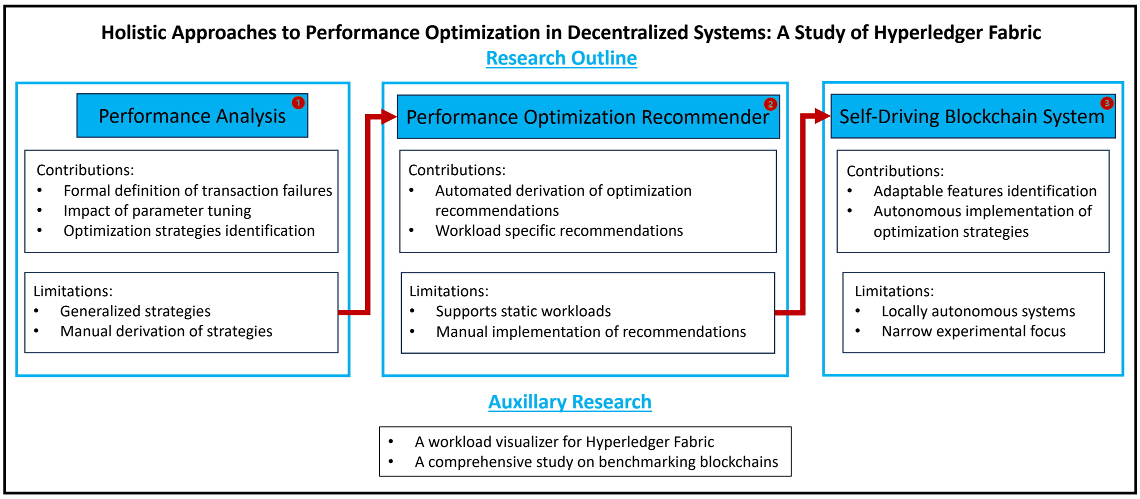

Holistic Approaches to Performance Optimization in Decentralized Systems: A Study of Hyperledger Fabric
Research Outline
Performance Analysis
1
Contributions:
• Formal definition of transaction failures
• Impact of parameter tuning
• Optimization strategies identification
Limitations:
• Generalized strategies
• Manual derivation of strategies
Performance Optimization Recommender
2
Contributions:
• Automated derivation of optimization recommendations
• Workload specific recommendations
Limitations:
• Supports static workloads
• Manual implementation of recommendations
Self-Driving Blockchain System
3
Contributions:
• Adaptable features identification
• Autonomous implementation of optimization strategies
Limitations:
• Locally autonomous systems
• Narrow experimental focus
Auxillary Research
• A workload visualizer for Hyperledger Fabric
• A comprehensive study on benchmarking blockchains

Figure 1.3.1: Outline of our research work

workload [36]. These fluctuations in performance necessitate the dynamic application of appropriate optimization strategies. However, manual dynamic tuning is a time-consuming and expensive process; hence, autonomous blockchain systems that can apply performance optimization strategies dynamically without human intervention are the need of the hour.

The existing literature on blockchain technology has begun to explore the potential for auto-tuning and self-adaptive systems, which can automatically adjust their parameters in response to changing conditions. However, this exploration has predominantly concentrated on singular facets of blockchain technology, such as tuning the configuration parameters, without taking a holistic view. Developing a fully comprehensive self-driving blockchain system presents a formidable challenge.

1.3 Approach

The diversity and complexity of blockchain technologies, with each blockchain having its own unique implementation and characteristics, pose a significant challenge to devising a one-size-fits-all solution to the various research problems in the field. This variability spans consensus mechanisms and transaction processing to smart contract functionality and governance models, making it difficult to develop universally applicable solutions. Fabric is renowned as one of the most popular permissioned blockchain networks, both in industrial applications and academic research, due to its distinctive features, such as its modular architecture, support for smart contracts in general-purpose languages, and its ability to ensure high levels of privacy and scalability [6, 37]. This makes it an ideal candidate for in-depth study and problem-solving within its framework. By concentrating on solving specific problems within Fabric's framework, we aim to create solutions that not only address these issues effectively within Fabric but also offer insights and methodologies that could be adapted to other decentralized systems.

An overview of our research is depicted in Figure 1.3.1 and shows a comprehensive strategy for performance enhancement, structured in three pivotal phases. The Figure demonstrates how each phase incrementally builds upon its predecessor, addressing and surpassing its constraints. Initially, our efforts were directed at benchmarking Fabric to pinpoint the primary causes of performance bottlenecks. The insights gleaned from our experiments enabled us to formulate broad yet incisive optimization strategies that enhance performance across various layers of the Fabric blockchain framework. Subsequently, the second phase of our research concentrated on automating the generation of optimization recommendations that are specifically tailored to the current workload. Manual implementation of these recommended strategies resulted in notable performance boosts. The third phase of our research focused on automating the implementation of optimization recommendations and adapting them dynamically based on workload changes. We investigated the feasibility of a self-driving blockchain system dedicated to continuous performance improvement. In addition to our main research, we also developed a Fabric workload visualizer to enhance our understanding of workload characteristics that impact performance. Furthermore, we conducted a study on benchmarking various blockchain systems to identify the primary challenges in this domain and to extend our research to other decentralized systems beyond Fabric.

1.3.1 Performance Analysis

Various types of transaction failures can occur in Fabric, but the ones related to concurrency are the most common. These failures can significantly impact Fabric's performance, as our studies show that almost 40% of submitted transactions can fail because of concurrency-related issues. Our work formally defines these types of transaction failures related to concurrency, laying a solid foundation for our study and future investigations in this area. We conducted over 900 experiments to analyze concurrency-related transaction failures in Fabric. By examining the factors that affect transaction failures in Fabric and the impact of three recent optimization techniques - Fabric++, Streamchain, and FabricSharp - we have found interesting insights pertaining to different levels of the blockchain stack. For instance, we discovered that adjusting the block size based on the rate of transaction arrivals can significantly reduce transaction failures by up to 60%. This highlights the importance of tuning configuration parameters to improve the performance of Fabric.

To support thorough evaluations and controlled experiments, we developed four smart contracts representing realistic scenarios along with a synthetic smart contract. Additionally, we have developed a workload generator that can be used for all of these smart contracts. This wide range of workloads has allowed us to conduct an extensive evaluation of Fabric. Based on our findings, we have identified some best practices that developers should follow while working with Fabric, and have also highlighted promising areas for future research.

We also developed a web application designed to analyze the workloads of a Hyperledger Fabric network and provide users with detailed insights into transaction metadata, dependencies and serializability. We demonstrated the tool's effectiveness using multiple realistic workloads, highlighting its utility in identifying optimization opportunities.

Further, we conducted a comprehensive study on benchmarking different blockchain systems. We analyzed five popular permissioned blockchain platforms to identify key problem statements and defined a general methodology to address these issues. We evaluated five existing blockchain benchmarking systems through detailed case studies, identifying their limitations and proposing enhancements based on our findings.

1.3.2 Performance Optimization Recommendation System

We designed a multi-level optimization recommendation approach that analyzes the blockchain ledger and identifies various optimization opportunities for the entire blockchain stack. This approach equips users with a detailed understanding of their systems, enabling them to make informed decisions on optimization strategies. Our approach also automates the extraction, preprocessing, and generation of event logs for Fabric blockchain data. This facilitates further research in log-based analysis techniques such as process mining in blockchains by providing a ready-to-use, preprocessed log based on the ledger contents. Through practical implementation and evaluation, we demonstrate that our optimization recommendations can significantly enhance transaction success rates by 20% and reduce latency by 40%, underscoring the effectiveness of our approach.

We conducted an extensive evaluation of our approach, which encompasses a variety of workloads, including synthetic workloads, use-case-based workloads and a real-world event log. This extensive testing ensures the applicability of our approach to a wide range of real-world blockchain applications, addressing the challenge of limited publicly available data for research in permissioned blockchains. Moreover, we demonstrate how our approach enhances existing blockchain optimization strategies by adding higher-level optimizations, thus confirming the significant benefits of our holistic approach.

1.3.3 Self-driving Blockchain System

Our research delves into the dynamic aspects of the Fabric blockchain ecosystem, with a focus on evolving network and workload patterns. We investigate the feasibility and potential benefits of self-driving blockchains. We compare our findings with established database research and highlight the unique attributes of blockchain technology. Our objective is to encourage the blockchain research community to contribute significantly to developing self-driving blockchains.

Our research has identified specific parameters and components within the entire system stack of Hyperledger Fabric that are well-suited for autonomous adaptation. These particular features, which we refer to as *adaptable features*, present significant potential for self-driving opportunities within existing blockchain systems. However, we recognize that there are considerable challenges in making these features fully autonomous and have provided potential solutions to overcome these obstacles. Our aim is to facilitate the exploration of self-driving capabilities within existing blockchain systems rather than the creation of new blockchains for each unique use case.

Finally, we demonstrate our findings by setting up three autonomous systems, each targeting a different level within the blockchain architecture. Our experiments indicate up to 42% improvement in terms of throughput and latency. This is a significant first step towards implementing a fully autonomous system in the future.

1.4 Contributions

Our research involves a thorough analysis of blockchain technology from different perspectives to identify areas that are causing performance issues. We then develop strategies that target these areas and optimize the entire blockchain stack. This includes improving the data definition, transaction processing, and even the business process model. By taking a holistic approach, we can pinpoint and address the main causes of inefficiencies, aiming to enhance the performance of blockchain systems. The main contributions of our work are:

i. We formally define different types of transaction failures related to concurrency that can occur in Fabric, which is helpful for our study and future research. We examine the parameters affecting these failures in Fabric and three other optimized extensions of Fabric. Our findings reveal surprising insights and trade-offs, such as the significant impact of block size on transaction failures. Based on our experiments, we identify best practices for developers and future research directions [24].

ii. We define a multi-level optimization recommendation approach that examines the Fabric blockchain ledger and recommends optimization strategies for the entire blockchain stack. Our work provides a formal definition for our optimization strategies that can be reused for other blockchain logs with similar attributes. The effectiveness of our method is demonstrated by the results, which show a 20% increase in successful transactions and a 40% reduction in latency. The approach we present supplements existing blockchain optimization strategies by adding higher-level optimization techniques [27].

iii. Our work analyzes the dynamic aspects of the blockchain environment to determine the need for self-driving blockchains. We identify adaptable features in the Hyperledger Fabric system stack that are suitable for autonomous adaptation and demonstrated significant improvement in performance through three locally autonomous systems. This is a significant first step towards implementing a fully autonomous system in the future, and the first comprehensive discussion and evaluation of self-driving blockchains to our knowledge at the time of writing [38].

iv. To understand the workload characteristics, we developed a web application that visualizes transaction dependency graphs and other key transaction-related information for Hyperledger Fabric networks. Our tool helps identify performance optimization opportunities relevant to the current workload [28].

v. We analyse five popular permissioned blockchains to identify key benchmarking challenges and propose a general methodology to address these issues. Additionally, we evaluate existing blockchain benchmarking systems, highlighting their limitations and suggesting improvements [39].

vi. To facilitate realistic evaluations and controlled experiments, we created a collection of synthetic workloads generated using a broad range of control variables, four popular use-case-based workloads, and a workload based on a real-world process event log. All the smart contracts and workload generation scripts have been made available as open-source [24]. This helps address the challenge of inadequate publicly available data, which hinders research in permissioned blockchains.

Parts of the content and contributions of this work have been published in:

- Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. "Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric." In: *Proceedings of the 2021 International Conference on Management of Data.* SIGMOD '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 221–234. ISBN: 9781450383431. DOI: 10.1145/3448016.3452823. URL: https://doi.org/10.1145/3448016.3452823 **(CORE PUBLICATION 1)**

- Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. “How To Optimize My Blockchain? A Multi-Level Recommendation Approach.” In: *Proc. ACM Manag. Data* 1.1 (May 2023). DOI: 10.1145/3588704. URL: https://doi.org/10.1145/3588704 (New publication format for SIGMOD ‘23) **(CORE PUBLICATION 2)**
- Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. *Should my Blockchain Learn to Drive? A Study of Hyperledger Fabric.* 2024. arXiv: 2406.06318
- Jeeta Ann Chacko, Ruben Mayer, Alan Fekete, Vincent Gramoli, and Hans-Arno Jacobsen. “A Comprehensive Study on Benchmarking Permissioned Blockchains.” In: *Performance Evaluation and Benchmarking.* Ed. by Raghunath Nambiar and Meikel Poess. Cham: Springer Nature Switzerland, 2024 *(Accepted for publication)*
- Jeeta Ann Chacko, Nino Richter, Ruben Mayer, and Hans-Arno Jacobsen. “Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric.” In: *Proceedings of the 24th International Middleware Conference: Demos, Posters and Doctoral Symposium.* New York, NY, USA: Association for Computing Machinery, 2023, pp. 31–32. ISBN: 9798400704291. URL: https://doi.org/10.1145/3626564.3629098

1.5 Organization

The rest of the document is organized as follows. In Chapter 2, we explain the Hyperledger Fabric architecture including the transaction flow to provide a background for our work. In Chapter 3, we present our methodology for benchmarking and optimization of the Hyperledger Fabric blockchain. This includes explaining our experimental setup, defining the workloads used in our experiments, and summarizing the system design. In Chapter 4, we provide a summary of the key achievements of each publication and the author’s contributions. Chapter 5 compares our findings with the existing literature and discusses the results. Chapter 6 presents the conclusion and an outlook for future work. Additionally, Appendices A, B, C, D, and E contain our published papers.

2

Background

Our work focuses on enhancing the overall performance of decentralized systems through comprehensive optimization strategies. We utilize the Hyperledger Fabric blockchain platform to showcase our techniques. To better understand our work, we explain the architecture and transaction flow within Fabric in this section.

Fabric [6] is an open-source permissioned decentralized system designed mainly for use in enterprise scenarios. It supports smart contracts (referred to as chaincodes) in various general-purpose languages. It also offers a choice of two databases, LevelDB and CouchDB, to maintain the world state in a key-value format. The complete history of all transactions (both successful and failed) in the network is stored on a distributed immutable ledger. The main components of a Fabric network are clients, peers, endorsers, and the ordering service. Peers are responsible for validating transactions, updating the world state and appending transactions to the ledger. Endorsers are specific peers assigned to execute transactions speculatively and endorse them. A copy of the smart contract is stored on every endorser. An endorsement policy governs the number of endorsements required for a transaction to be deemed valid. Peers are grouped into organizations that generally correspond to the real organizations in an enterprise. The ordering service is a set of nodes that reach consensus on a global order of transactions and package them into blocks. The transaction flow in Fabric is shown in Figure 2.0.1 and is explained below in detail.

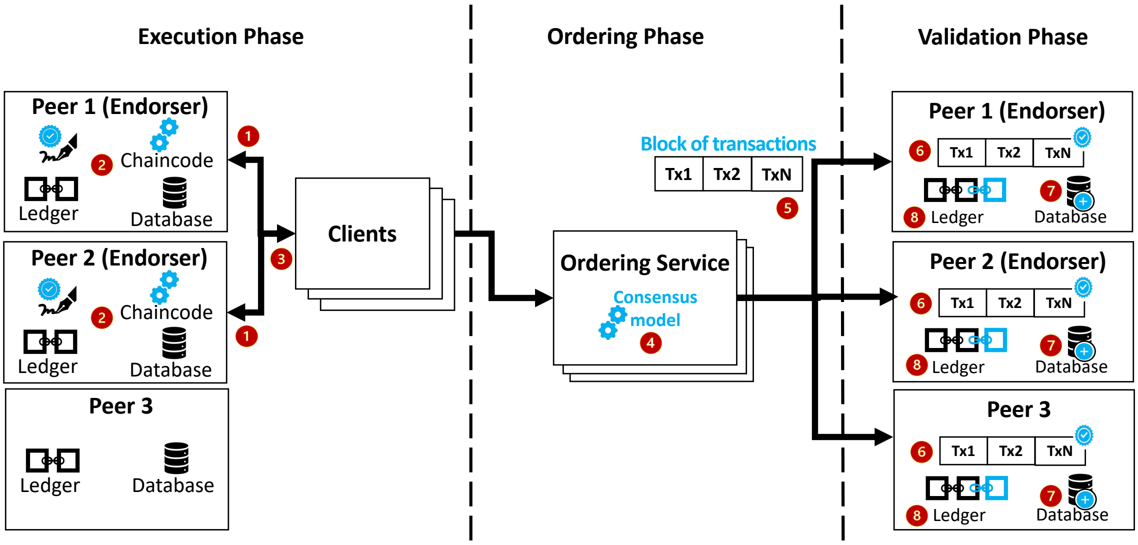


Figure 2.0.1: Transaction Flow in Hyperledger Fabric

Execution Phase

1. The client can initiate a transaction by sending a transaction proposal to all endorsers specified by the endorsement policy. For instance, if the endorsement policy is *(Peer1 AND Peer2)*, the clients send proposals to both *Peer1* and *Peer2*. The transaction may comprise of several operations that involve reading from or writing to the world state.
2. The endorsers execute the transaction speculatively and generate a read/write set based on the current world state. The world state remains unchanged during this process. The read/write set is signed by the endorsers and sent back to the clients.
3. Clients collect sufficient responses from endorsers based on the endorsement policy. The responses are packaged as a transaction and sent to the ordering service.

Ordering Phase

4. The ordering service nodes reach consensus on a global order of transactions and create a block. The frequency of block creation can be based on a time duration, number of transactions or size of transactions.
5. After the transactions are grouped together in a block, they are distributed to every peer in the network.

Validation Phase

6. Peers perform two validation processes on every transaction in a block. Validation System Chaincode (VSCC) validation involves validating the endorser signatures and ensuring sufficient endorsements based on the endorsement policy. Multi-Version Concurrency Control (MVCC) validation involves comparing the read/write set with the current world state. MVCC validation is required because the read/write sets were created based on the world state during the execution phase, and other transactions might have updated the world state in the meantime.
7. When both the VSCC and MVCC validations are successful, the world state is promptly updated. However, if either of these validations fails, the clients are immediately notified, and the world state remains unchanged.
8. Once validation is complete, the block is updated with the status of each transaction (whether it was successful or failed) and appended to the ledger. As a result, the ledger holds a complete record of all transactions that have completed the transaction flow.

Fabric provides multiple customizable options, such as allowing for diverse ledger data storage formats and different databases for world state storage. Users have the choice between crash fault tolerant or byzantine fault tolerant consensus models, allowing them a tradeoff between performance and security. It supports various membership service providers and enables smart contract development in multiple programming languages, including Go, Node.js, and Java. Additionally, various configurable settings are available in Fabric to maintain transaction privacy, such as the creation of channels. Channels enable a subset of network participants to maintain a private transaction ledger, which is desirable for scenarios where participants are competitors and wish to keep certain transactions confidential. Fabric further enhances privacy with the feature of private data collections, allowing a specific group within a channel to handle private data without needing a separate channel. This feature can be used to store private information related to an individual organization, such as the details of employees or assets. Overall, Fabric is a highly flexible framework with a range of features that make it suitable for enterprise use cases.

3

Methodology

In our work, we first conduct a thorough examination of Fabric's performance to identify any potential bottlenecks. We then analyze the data in the distributed ledger to derive optimization strategies at different levels of the blockchain stack. In addition, we develop a comprehensive optimization recommendation system as well as a self-driving blockchain framework that applies these strategies to enhance overall performance.

For our initial analysis of Fabric and to evaluate the two systems that we developed, we deployed a distributed Fabric network on our servers and designed a suitable benchmarking setup. This section outlines our general methodology, which includes the Fabric system setup, the smart contracts and workloads used in our experiments, the multi-level optimization strategies that we identified and the design of our two holistic performance optimization systems.

Although they are three independent systems, we illustrate the design of the Fabric network, the workflow of our recommendation system, and the components for the self-driving framework together in a single figure for ease of understanding (Figure 3.1.1). Independent illustrations of the system design can be found in our papers in appendices A, B, and C. Further, in this section, we briefly explain our auxiliary research, which includes the development of a Fabric workload visualizer and a study on blockchain benchmarking systems. Detailed information can be found in appendices D and E.

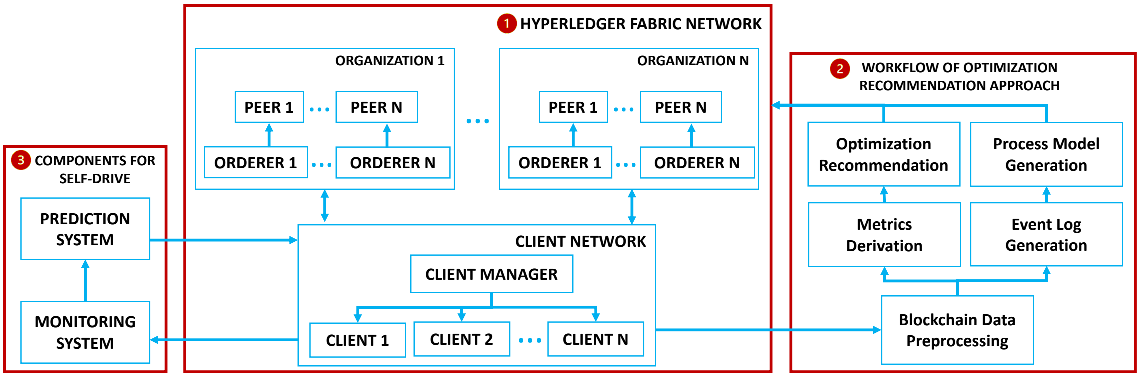


Figure 3.1.1: System Design of the Fabric Network along with the workflow of the holistic performance optimization approach and components for a self-driving blockchain system

3.1 Fabric Network Setup and Analysis

As depicted in Figure 3.1.1, the Fabric network consists of multiple peers and orderers grouped into different organizations, which simulate the physical organizations of an enterprise. Clients controlled by a client manager are registered to this Fabric network, which generates configurable workloads for multiple smart contracts. The default Fabric network configuration included four peers (two per organization) and three orderers along with ten clients and a client manager deployed using Hyperledger Caliper which is a benchmarking framework [40]. This default configuration was tuned based on the requirements for each experiment. We used Fabric versions 1.4 and 2.2 for our experiments.

The Fabric network was deployed over a Kubernetes cluster. The various nodes in the cluster consist of command line interface nodes (CLI), controller nodes, worker nodes, a load balancer, and a network file system node. The CLI nodes facilitate client processes, while the controller nodes manage scheduling within the Kubernetes environment. Worker nodes hosted the Fabric network components, including peers and orderers. The use of Kubernetes as the orchestration platform enabled a flexible and scalable deployment model. The setups varied in scale from small clusters with three worker nodes to larger configurations with 32 worker nodes, accommodating different numbers of peers, orderers, and client processes. This variation allowed for the exploration of network behaviour under varying degrees of workload and stress. Hardware configurations varied, with nodes equipped with up to 16 vCPUs and 41 GB RAM.

3.1.1 Smart Contracts and Workloads

Efficiently benchmarking the performance of Fabric and testing the systems we've developed requires a diverse range of workloads. In blockchains, smart contract executions are equivalent to transactions in a database, making it imperative to create appropriate smart contracts for generating the necessary workloads. In this section, the smart contracts and corresponding workloads that we developed for our experiments are explained.

Synthetic Workloads

We developed a workload generator to run controlled experiments and microbenchmarks. It generates synthetic smart contracts based on user-specified control variables, such as the number of functions and operations. The generator can also produce synthetic workloads based on inputs like transaction numbers and key distributions. We create diverse workloads to simulate realistic scenarios, including read-heavy, insert-heavy, update-heavy, delete-heavy, and range-heavy workloads.

Use-case based Workloads

Blockchains are being used in various real-world scenarios, including digital asset and supply chain management [12, 13]. To create a more realistic environment for our experiments, we implemented four smart contracts and corresponding workloads based on popular use cases of blockchains. The design and functionality of these use cases draw inspiration from similar implementations in the existing literature [41, 42, 43, 44].

1. **Electronic Health Record Management**
This use case manages medical health records efficiently, catering to the needs of patients, medical institutions, and research organizations. Patients can control access to their personal information and health records. The smart contract manages access credentials

and logical connections between patients and medical actors. The workload generated for testing this smart contract is predominantly update-heavy, comprising 70% of the transactions in a set of 10,000.

2. **Digital Voting**
This use case simulates a secure, transparent digital election with 1,000 eligible voters and 12 parties. The voting occurs in a designated election phase, ending with a *closeElection* transaction and ensures that no vote duplication occurs. The system includes functions for querying party information (*qryParties*), casting votes (*vote*), and viewing results (*seeResults*), ensuring transparency. The corresponding workload reflects election traffic with 1,000 *qryParties* transactions at 100 TPS and 5,000 *Vote* transactions at 300 TPS, ending with result viewing and election closure.

3. **Supply Chain Management**
This use case simulates logistics operations of a network, tracking and managing logistic units across five logistic service providers. The system utilizes global trade item numbers and serial shipping container codes to monitor individual trade items and groups of items throughout the supply chain. A workload of 10,000 transactions is generated to simulate realistic supply chain activities, including the sending of advanced shipping notices, shipping, querying, and unloading transactions, as well as random transactions for querying product information and updating audit entries. This use case replicates the intricacies of a logistics network, from tracking and management to detailed audit and information retrieval processes.

4. **Digital Rights Management**
This use case employs blockchain technology to provide a secure platform for managing and protecting the works of artists in the music industry. The smart contract leverages the dot blockchain media format to store metadata for 200 artworks and assign industry-standard IDs to 200 right holders. The comprehensive functionality of this smart contract includes adding new music pieces, querying rights associated with each piece, viewing detailed metadata, calculating the current revenue for rights holders, and a *Play* function triggered with each music play. A *Play* heavy workload is created to simulate realistic usage scenarios with 70% of transactions dedicated to the *Play* function.

3.1.2 Performance Analysis

Using the smart contracts and workloads explained in the previous section, we conducted over 900 experiments on our Fabric network setup. We observe that in Fabric, a significant proportion of transaction failures, nearly 40%, are attributed to concurrency issues, which markedly influence the system's efficiency. Our work provides a comprehensive categorization and definition of these concurrency-related transaction failures, establishing a foundational basis for this study and future research endeavours. Our analysis extended to the effects of various factors across different layers of the blockchain stack, such as the endorsement policy, database type, network delays and workload type, on transaction failures. One key finding was the potential reduction in transaction failures, up to 60%, by adapting the block size in alignment with transaction arrival rates. We also conducted experiments on optimized extensions of Fabric namely *Fabric++* [25], *Streamchain* [34], and *FabricSharp* [26] and derived insightful observations.

3.1.3 Workload Visualizer

As a result of our performance analysis, we determined that workload characteristics significantly impact performance and that optimal configuration settings are also workload-dependent. Consequently, we developed a web application named Fabric-Visualizer to facilitate the visualization of workloads within Hyperledger Fabric networks. Our methodology involved connecting to a live Fabric network to extract transaction data and generate dependency graphs. These graphs offer detailed insights into validation status, dependent transactions, accessed keys, endorsers, and clients. Furthermore, our tool checks for serializability and identifies transactions that require abortion to achieve serializability. We validated the utility of Fabric-Visualizer by employing multiple realistic workloads, demonstrating its effectiveness in identifying performance optimization opportunities.

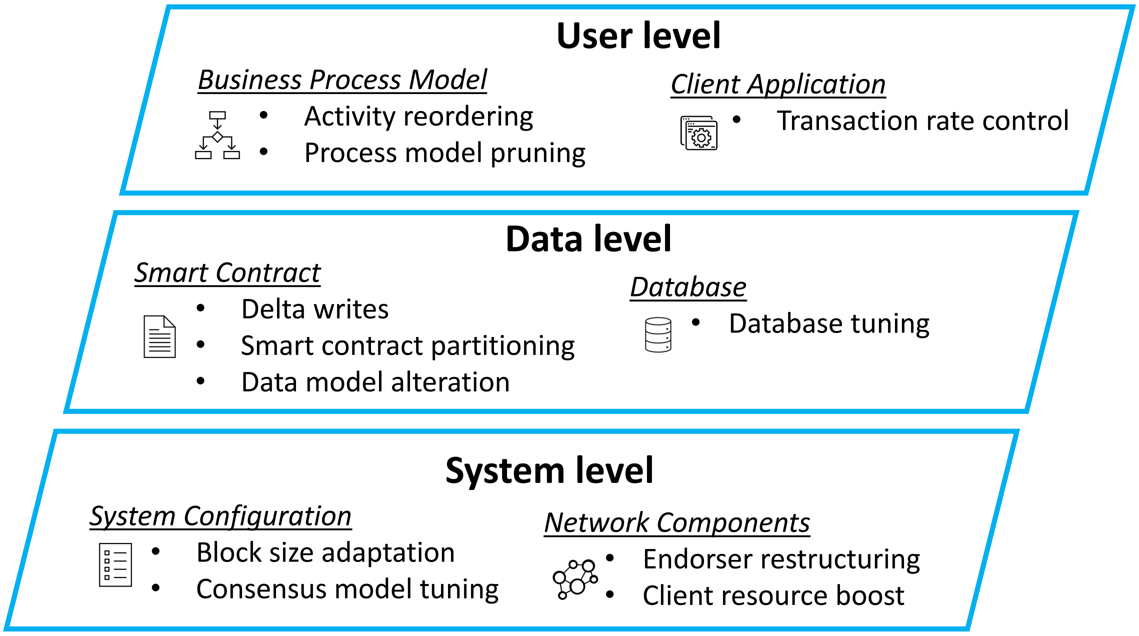


Figure 3.2.1: Performance optimizations applicable to different elements of the Fabric blockchain stack

3.2 Holistic Performance Optimization Strategies

On the basis of our extensive experiments explained in the previous section, further analysis of the data in the distributed ledger and literature review, we identified various optimization possibilities at different levels of the Fabric blockchain stack. These optimization strategies are visualized in Figure 3.2.1 and explained in this section.

3.2.1 User Level Optimizations

At the user level, we investigated the workload of the active application. The performance of the system is significantly impacted by the frequency and sequence in which transactions are initiated and authenticated on the blockchain. A detailed analysis of the transaction rate, type, and interdependencies can help identify optimization opportunities for the user.

Using the data in the distributed ledger, we can identify reorderable pairs of transactions that cause transaction dependency failures and the business process activities associated with such pairs. An appropriate process model redesign by reordering the identified activities can mitigate transaction dependency conflicts.

Further, when activities stray from their expected patterns, it's advisable to consider pruning the process model. This involves analyzing the transaction types associated with each activity to spot any irregularities. By examining the discrepancies between the event log's traces and the process model, alongside the anomalies identified, one can pinpoint opportunities for refining the model.

We can also analyze the distribution of transaction rates over time from the distributed ledger, identifying moments of peak activity and concurrently assessing failure rates within those periods. If the failure rate is significantly elevated, implementing rate control measures can help improve performance. Further, adjusting the rate at which transactions are admitted into the system can maintain fairness among participants.

3.2.2 Data Level Optimizations

At the data level, we examine aspects such as transaction failures, correlation based on proximity, and hotkeys. Such analysis enables users to modify the smart contract and, consequently, the foundational data model, leading to enhanced performance.

One approach, specifically for transactions limited to incrementing or decrementing operations, is to transform them into delta-write transactions. These allow for modifications across multiple distinct delta keys, which can then be combined to obtain the current value when needed. This eliminates the need to read the key prior to each modification. As a result, update transactions morph into write-only operations targeting unique keys, significantly lowering the likelihood of failures due to transaction dependencies. Delta writes are particularly advisable in scenarios where a transaction fails while attempting to increment or decrement a single key.

To mitigate transaction dependencies, one approach is to divide a single smart contract into multiple ones, each interacting with its own distinct world state to minimize conflicts.

This division does not alter the original smart contract's functionality, as functions from one smart contract can still call upon those in another when interaction is necessary. Analyzing and modifying the smart contract accordingly to facilitate this split — termed smart contract partitioning — is advised when a single key becomes a bottleneck due to concurrent access by multiple functions.

Further, when activities are self-dependent, modifying the data model can help in minimizing transaction conflicts. For instance, in the context of digital voting, if an *ElectionID* key becomes overused due to frequent access by the *Vote()* function, an effective optimization strategy could be to shift the primary key to something like *VoterID*. This change allows for votes to be recorded on a per-voter basis, rather than compiling all votes under a single key. Additionally, the discovery of a singular *hot key* warrants a thorough examination of the data model to uncover why this particular key is disproportionately accessed.

Further, in Fabric, the different databases have diverse performance and functionality suitable for various workloads. Therefore, choosing between LevelDB and CouchDB based on the workload profile can improve the performance. Also, dynamically tuning the various database parameters can ensure higher efficiency.

3.2.3 System Level Optimizations

At the system level, our emphasis lies on various configuration parameters that can particularly influence its performance. Additionally, identifying client and peer bottlenecks assist in making informed resource allocation choices. These optimization strategies should be based on analysis of the blockchain ledger produced by the active application, enabling users to determine the most suitable configuration settings tailored to their specific scenario and workload, thereby enhancing performance.

Max Message Count determines the maximum number of transactions a block can contain. *Batch Timeout* is the maximum duration to wait before creating a block with the currently available transactions. *Preferred Max Bytes* specifies the maximum block size in terms of bytes. Tuning these three parameters can optimize transaction processing efficiency and impact network throughput and latency. In the context of the consensus mechanism (such as Raft), the *Snapshot Interval Size* parameter defines the frequency of state snapshotting, which can affect recovery times and disk space utilization. Further, Fabric provides a flexible consensus framework that can be configured to suit the trust model and behavior of the network participants. The network performance can be improved by switching between different consensus mechanisms, such as CFT and BFT.

Additionally, adjusting the number and distribution of network components, such as endorsers and clients, in response to changing workloads can help to maintain optimal performance. In Fabric, each transaction initiated by clients triggers the execution of a specific smart contract function by endorsers, as determined by the endorsement policy. This process, which is both time-intensive and resource-demanding, can lead to inefficiencies if certain endorsers are overloaded with transactions while others are underutilized, indicating a potential bottleneck or imbalance in load distribution. This imbalance often arises from endorsement policies that require mandatory endorsements from certain endorsers, leading to a scenario where specific endorsers, like *Org1* in the policy *And(Org1,OR(Org2,Org3))*, become overwhelmed and turn into bottlenecks.

Similarly, clients undertake various labour-intensive operations, encompassing transaction proposal initiation, validation of endorser responses, a compilation of these responses into a transaction, submission of the transaction to the ordering service, and gathering of commit responses from peers. Client bottlenecks can be detected by identifying the clients responsible for a significant volume of transactions.

Recognizing such over-utilized clients and endorsers aids in making informed decisions about resource distribution, for instance, augmenting the quantity and capacity of these components. Additionally, such identification may highlight inefficiencies or issues within the foundational business processes.

3.3 Holistic Performance Optimization Systems

In our work, we devised two systems that cater to the holistic approach of enhancing the performance of Fabric. Firstly, we developed a performance optimization recommender system that identifies inefficiencies and bottlenecks in the blockchain's operation, generating insights into potential optimizations at various levels (system, data, and user). The approach is structured around data preprocessing, event log extraction, process analysis, and the formulation of optimization strategies. Secondly, we develop a self-driving blockchain system that utilizes machine learning algorithms to dynamically adjust blockchain parameters in real-time, in an attempt to ensure optimal performance under varying conditions. This involves monitoring system metrics, analyzing performance data, and applying reinforcement learning to identify and implement the best configurations autonomously. Both systems are designed with a keen focus on providing a comprehensive solution to the challenges faced by Fabric. In this section, we will delve into the details of both these systems.

3.3.1 Performance Optimization Recommendation System

We propose a transaction-centric approach to derive optimization recommendations for blockchains by utilizing the data in the distributed ledger. We preprocess the raw data to create a blockchain log and apply process mining strategies to derive the process model. Further, we derive the values for various metrics from the blockchain log, which are used to detect multi-level optimization recommendations. We then use the recommendations and the derived process model to identify applicable optimizations. To automate this approach, we have developed a tool called *BlockOptR*. The workflow of our system is illustrated in Figure 3.1.1.

BlockOptR is registered as a client on the Fabric network to extract data from the blockchain ledger. This log is then saved as JSON files and cleaned by removing configuration and setup-related transactions. This process produces a CSV format file that is used as an event log by generating appropriate Case IDs. Process mining techniques are applied to the event log to derive process models and aid in recommending user-level optimizations. Further, nine attributes are extracted from the blockchain log, which helps derive multiple metrics required to recommend optimizations. Using a multi-level approach, the defined attributes and metrics are utilized to recommend nine performance optimization strategies, namely activity reordering, process model pruning, transaction rate control, delta writes, smart contract partitioning, data model alteration, block size adaptation, endorser restructuring and client resource boost.

3.3.2 Self-Driving Blockchain System

Our work explores the development of self-driving blockchains, focusing on enhancing Fabric's adaptability and efficiency. We delve into the concept of autonomously adjusting blockchain parameters and configurations in response to changing network conditions, workloads, and performance metrics. Through a comprehensive analysis of various layers of the blockchain stack—ranging from system and data layers to the application layer—our work proposes a novel framework that leverages reinforcement learning to dynamically tune parameters. Based on the optimization strategies that we derived (Section 3.2), further experimentation and literature review, we identified various adaptable features across the blockchain stack that are integral to realizing a self-driving blockchain system.

These adaptable features represent areas for dynamic adjustment to improve blockchain system performance, responsiveness to changing workloads, and overall system efficiency. We highlight the importance of a self-driving approach to manage these adaptations autonomously, leveraging machine learning to predict and implement optimal configurations in real time. Further, we developed three self-driving systems, each targeting different layers of the Fabric blockchain. At the user level, we apply dynamic transaction rate control with a focus on client fairness. At the data level, we dynamically adapt the smart contract to include or exclude *delta writes* based on workload changes. At the system level, we dynamically adapt four configuration parameters to ensure better performance. The workflow of our system is illustrated in Figure 3.1.1.

Our autonomous blockchain system has two main components: a prediction system and a monitoring system, both integrated with a Fabric network. The prediction system uses reinforcement learning (RL) to predict suitable optimizations and manipulate Fabric's adaptable features. Reinforcement learning is especially useful in situations where initial training data isn't available. Given the privacy concerns of enterprises reluctant to disclose their operational data and ledger details, permissioned blockchains lack extensive public training datasets. This scenario makes reinforcement learning an apt method for our application. The monitoring system collects performance data from the network. The state, action space, and reward function parameters of the RL agent are adjusted to implement self-driving capabilities at each level of the blockchain stack. Initial results indicate promising performance improvements, which can be further enhanced by refining the training model.

3.4 Blockchain Benchmarking Systems

We extensively study multiple blockchain systems and blockchain benchmarking systems to broaden the scope of our research beyond Hyperledger Fabric. We address the significant challenges in benchmarking different blockchains which arise from their diverse implementations. To identify critical benchmarking challenges, we performed an in-depth analysis of five prominent permissioned blockchain platforms: Fabric, Corda, Multichain, Quorum, and Diem. Our examination encompassed system configuration, parameter tuning, workloads, and performance metrics, allowing us to formulate specific problem statements for each aspect. Subsequently, we proposed a comprehensive methodology to address these challenges, highlighting the essential contributions required from both blockchain and benchmarking system developers. Additionally, we conducted a case study on five existing blockchain benchmarking systems - Blockbench, Caliper, Diablo, Gromit, and BCTMark - evaluating their limitations and proposing enhancements. Our research aims to refine the benchmarking process and enable more accurate and meaningful performance evaluations across various blockchain platforms.

4

Summary of Publications

This chapter summarizes all published papers for this publication-based dissertation individually. This dissertation is based on four peer-reviewed publications (two full papers, one workshop paper and one demo paper) and one arXiv publication. We highlight each publication's key idea, outline the achievements, and summarize the author's contributions.

Section 4.1 outlines our first publication that focuses on identifying critical factors affecting transaction failures in the Hyperledger Fabric platform through extensive benchmarking experiments. We analyzed network configuration, transaction size, and endorsement policies, and compared existing optimization strategies. Section 4.2 summarizes our second publication that proposes an approach to optimize blockchain systems using a comprehensive framework backed by practical evaluations. The work emphasizes the importance of adopting a holistic approach to blockchain optimization. Section 4.3 abstracts our discussion on self-driving blockchains, focusing on Hyperledger Fabric. The study identifies crucial dynamically adaptable features and demonstrates the improvements in throughput and latency by implementing autonomous systems targeting different layers of the blockchain stack. Section 4.4 abstracts our extensive study on blockchain benchmarking systems, and Section 4.5 summarizes our work on the Fabric workload visualizer tool.

4.1 Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric

Reference: Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. "Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric." In: *Proceedings of the 2021 International Conference on Management of Data.* SIGMOD '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 221–234. ISBN: 9781450383431. DOI: 10.1145/3448016.3452823. URL: https://doi.org/10.1145/3448016.3452823 **(CORE PUBLICATION 1)**

Full-text version enclosed: Appendix A

Summary:

Our work thoroughly examines the Hyperledger Fabric platform and formally defines the different types of transaction failures that can occur in Fabric. We conducted over 900 experiments to identify key factors that lead to transaction failures.

Our work identifies several critical parameters affecting transaction failures through empirical research, including network configuration, transaction size, and endorsement policies. We also compared existing optimization strategies—Fabric++, Streamchain, and FabricSharp—highlighting their strengths and weaknesses in reducing transaction failures. We meticulously analyzed our experimental results to derive valuable insights into the complex dynamics of blockchain transaction processing.

Our work contributes significantly to the blockchain community by providing a deeper understanding of the causes of transaction failures in Hyperledger Fabric and suggesting practical solutions for improvement. We not only shed light on the technical challenges but also open avenues for future research and development in blockchain technology optimization. The recommendations made are poised to assist developers and researchers in enhancing the robustness and efficiency of blockchain systems.

Author's contributions: Conceived and developed the approach. Conducted analysis and experimental evaluation. Wrote the paper.

4.2 How To Optimize My Blockchain? A Multi-Level Recommendation Approach

Reference: Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. "How To Optimize My Blockchain? A Multi-Level Recommendation Approach." In: *Proc. ACM Manag. Data* 1.1 (May 2023). DOI: 10.1145/3588704. URL: https://doi.org/10.1145/3588704 (New publication format for SIGMOD '23) **(CORE PUBLICATION 2)**

Full-text version enclosed: Appendix B

Summary:

We explore an innovative approach to optimizing blockchain systems to address the critical challenges that impede their efficiency and scalability. We introduce a comprehensive optimization framework that targets multiple aspects of blockchain technology, including the application, data model and system components. We systematically parse the blockchain ledger to identify performance bottlenecks and suggest targeted optimizations. This framework is not only theoretical but also backed by practical evaluations, demonstrating its potential to enhance blockchain performance significantly.

Central to our work is the development and application of an optimization tool that automates the process of analyzing the blockchain ledger in Fabric and generating recommendations for performance improvements. Through detailed log analysis and performance metrics evaluation, this tool identifies inefficiencies and provides actionable insights for optimizing blockchain configurations. The experimental results presented in our work showcase the tool's effectiveness in various scenarios, highlighting its capacity to improve success rate by up to 20% and reduce latency by up to 40% against varying workloads. Our work underscores the significance of adopting a holistic approach to blockchain optimization.

Author's contributions: Conceived, developed, and implemented the approach. Conducted analysis and experimental evaluation. Wrote the paper.

4.3 Should my Blockchain Learn to Drive? A Study of Hyperledger Fabric

Reference: Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. *Should my Blockchain Learn to Drive? A Study of Hyperledger Fabric.* 2024. arXiv: 2406.06318

Full-text version enclosed: Appendix C

Summary:

Our work presents a comprehensive study on the implementation and evaluation of self-driving blockchains, focusing on Hyperledger Fabric. We explore the concept of autonomous blockchains that can predict workload changes and reconfigure themselves for optimal performance without human intervention. Our study demonstrates significant improvements in throughput and latency by implementing three locally autonomous systems targeting different layers of the blockchain stack, suggesting a promising direction towards fully autonomous blockchain systems.

Our research highlights the need for self-driving blockchains to adapt to evolving workloads and network conditions and ensure fairness across clients. It identifies specific parameters and components within Fabric that are suitable for autonomous adaptation and evaluates the feasibility of such adaptations through experiments. These adaptations include dynamic parameter tuning, smart contract adaptation, and client admission rate tuning, leading to improved system performance and fairness.

Self-driving blockchains represent a significant advancement in blockchain technology, offering potential solutions to the challenges of managing complex and dynamic blockchain systems. The experiments conducted provide evidence of the benefits of autonomous adaptations on performance. Our research opens up new possibilities for the development of fully autonomous blockchain systems.

Author's contributions: Conceived, developed, and implemented the approach. Conducted analysis and experimental evaluation. Wrote the paper.

4.4 A Comprehensive Study on Benchmarking Permissioned Blockchains

Reference: Jeeta Ann Chacko, Ruben Mayer, Alan Fekete, Vincent Gramoli, and Hans-Arno Jacobsen. “A Comprehensive Study on Benchmarking Permissioned Blockchains.” In: *Performance Evaluation and Benchmarking*. Ed. by Raghunath Nambiar and Meikel Poess. Cham: Springer Nature Switzerland, 2024 *(Accepted for publication)*

Full-text version enclosed: Appendix D

Summary:

Our work addresses the significant challenges in benchmarking various permissioned blockchain systems due to their diverse implementations and rapid market growth. We emphasize the importance of establishing a standardized methodology for evaluating these blockchains, which are increasingly used in enterprise applications such as banking, supply chain transparency, and digital asset management. Our study reviews five popular permissioned blockchains—Hyperledger Fabric, Corda, Multichain, Quorum, and Diem—highlighting their unique characteristics and the necessity for a comprehensive benchmarking approach to assist developers and users in making informed decisions.

We identify key problems across four benchmarking factors: system configuration, parameter tuning, workloads, and performance metrics. We propose a detailed methodology to address these issues, involving contributions from both blockchain and benchmarking system developers. Our study includes a case analysis of five existing blockchain benchmarking systems, pointing out their limitations and suggesting improvements. Our findings stress the need for detailed documentation, workload-specific benchmarks, and fine-grained performance metrics to ensure fair and effective benchmarking. There is a dire need for active engagement from the blockchain community to enhance the benchmarking process, thereby facilitating the optimal selection and configuration of blockchain systems.

Author’s contributions: Conceived the idea and conducted the study. Wrote the paper.

4.5 Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric

Reference: Jeeta Ann Chacko, Nino Richter, Ruben Mayer, and Hans-Arno Jacobsen. "Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric." In: *Proceedings of the 24th International Middleware Conference: Demos, Posters and Doctoral Symposium.* New York, NY, USA: Association for Computing Machinery, 2023, pp. 31–32. ISBN: 9798400704291. URL: https://doi.org/10.1145/3626564.3629098

Full-text version enclosed: Appendix E

Summary:

We developed a web application designed to analyze transaction dependencies within Hyperledger Fabric workloads. The Fabric-Visualizer tool extracts data from a live Fabric network, generates transaction dependency graphs, and provides detailed information on each transaction, such as validation status, dependent transactions, smart contract functions, accessed keys, endorsers, and clients. This comprehensive visualization aids users in identifying the causes of transaction failures and optimizing smart contracts. The tool also checks for the serializability of transactions and identifies which transactions need to be aborted to achieve it, thereby helping users select appropriate optimization strategies.

The system is demonstrated with multiple realistic workloads and showcases its ability to provide valuable insights into transaction conflicts. The generated conflict graphs and detailed transaction information help users understand the impact of different workloads on transaction failures and serializability. Additionally, by analyzing transaction details, users can detect smart contract functions and keys causing conflicts, enabling them to implement optimization strategies like smart contract partitioning and activity pruning. Our tool assists users to better understand their blockchain workloads and make informed decisions on optimization strategies.

Author's contributions: Conceived the idea and conducted experimental evaluation. Wrote the paper.

5

Discussion

This chapter discusses our work in the larger context of performance optimization strategies in transaction processing systems. We present related work from the literature and highlight the specific aspects unique to our work. We conclude that a comprehensive approach to performance optimization that analyzes the entire system stack is highly beneficial for decentralized systems. We also discuss the technology independence of our general methodology by highlighting examples from other blockchain systems that are comparable to Fabric.

Our research advances the understanding of transaction failures in Fabric by defining failure types and examining the effects of key parameters on performance. Unlike previous studies that mainly focus on throughput and latency [32, 45], our analysis includes a broader comparison across several systems (Fabric 1.4, Fabric++, Streamchain, FabricSharp) under various workloads, with a special emphasis on transaction failures. We reveal insights not covered in existing literature [32], such as the impact of CouchDB on latency and the cost of range queries. We also explore how different Fabric versions and configurations influence transaction failures, offering new findings like the inverse relationship between block size and transaction failures at fixed rates across different versions of Fabric, which are not discussed in the related work.

Additionally, our work compares with database benchmarking research [46, 47] that focuses on identifying performance bottlenecks. However, the distinctive features of Fabric and blockchains in general, such as smart contracts, endorsement policies and consensus protocols, yield unique insights into managing transaction failures that are not discussed in database-related literature. Our comprehensive experimental analysis fills gaps left by previous studies and provides valuable insights for Fabric users.

Our research introduces a novel approach to enhancing blockchain performance through a recommender system that analyzes the data in a Fabric blockchain ledger and offers nine different strategies for optimizing the entire blockchain stack. The existing literature focuses on individual aspects of performance optimizations, such as transaction reordering, block size adjustments, CRDTs, and component parallelization, while we discuss a more comprehensive approach [26, 33, 34, 48]. Although we draw inspiration from database optimization and recommendation research [49, 50, 51], our focus is on blockchain-specific enhancements, considering unique factors like block size, endorsement policies, and smart contracts.

Additionally, we explore the use of process mining techniques on the data in the blockchain ledger to derive process-level insights that can improve the blockchain system's performance. This marks a departure from existing studies [52, 53, 54, 55, 56], which primarily focus on process mining techniques only to derive and understand the business process model. Our work leverages a business process perspective by extracting process data from the blockchain ledger, analyzing blockchain-specific attributes and deriving performance optimization strategies.

Our research on autonomous blockchains highlights the importance of self-managing systems and outlines the essential features needing continuous adjustment. Previous studies in database management have explored self-driving, auto-tuning, and adaptive systems across various aspects like resource allocation and query optimization [57, 58, 59, 60]. However, our focus shifts to distributed ledger technologies, which stand apart in architecture and functionality. This distinction also extends to exploring self-driving capabilities at the application level, particularly emphasizing fairness in decentralized systems like blockchains.

Recent studies have introduced self-adaptive and auto-tuning blockchains, such as Sabine [61], Ursa [62], Adachain [63], and Athena [64], each offering adaptability for a single aspect within the blockchain architecture [61, 62, 63, 64, 65, 66]. Our work, in contrast, seeks to identify and examine a comprehensive set of adaptable features across the entire blockchain stack. Further, unlike the related work that requires a restart of the blockchain system in order to apply dynamic adaptations [64], we focus on parameters that can be tuned on a live network.

We have demonstrated our comprehensive benchmarking and optimization techniques using the Fabric blockchain. Achieving technology independence can be difficult due to the significant variations in the implementation of different blockchain systems. However, we have identified specific examples that can serve as a roadmap for future researchers to apply our techniques to other blockchain systems.

At the system level, the literature discusses the impact of parameter tuning in various blockchain systems [67, 68]. For instance, in Quorum, the frequency of block time has a direct proportional impact on the transaction latencies. Transactions are verified by specific nodes called notaries in Corda, and configuring an ideal number of notaries based on the incoming workload can enhance the performance. The performance impact of these parameters is similar to the Fabric parameters that we discuss in our work, such as block size and endorsement policy. Further, all blockchain systems have dynamically tunable parameters (Corda and Multichain have more than 9 and 15 dynamic parameters, respectively), highlighting the need for self-drive in other blockchains.

Additionally, related work discusses several optimization strategies to reduce the gas fee and detect vulnerabilities for Solidity smart contracts, which align with our data-level recommendations. Various tools, such as Lorikeet and Caterpillar, make it possible to automate the creation and execution of Solidity smart contracts directly from process models. This provides a foundation for implementing process model-based optimizations similar to the ones proposed in our work on other blockchains. Further, client fairness is a universal concept that can be applied to all blockchain systems. Hence, strategies for application-level optimization, such as transaction admission rate control and client resource management, are not dependent on the blockchain platform.

6

Conclusions

Our work offers a thorough investigation into the optimization and management of decentralized systems with a focus on Hyperledger Fabric. Collectively, our three projects cover a broad spectrum of challenges and solutions concerning transaction failures, system performance, and the implementation of autonomous operations within blockchain systems.

Our experimental analysis of Fabric identifies the root causes of transaction failures, attributing them to its optimistic concurrency control mechanism. Through the development of diverse workloads, our study provides an exhaustive analysis of how various configurations impact transaction success rates. We also evaluate several optimization strategies proposed in the literature and offer actionable recommendations for practitioners to enhance system performance.

We further extend the discourse by proposing a multi-level optimization framework that encompasses system, data, and user-level adjustments. This approach is validated by applying the framework's recommendations, demonstrating substantial improvements in transaction throughput and latency. This research underscores the importance of a comprehensive optimization strategy that addresses multiple facets of blockchain systems to achieve significant performance gains.

Eventually, our focus shifts towards the concept of self-driving blockchains, which autonomously adapt to changing workloads and network conditions to optimize performance metrics such as throughput, fairness, and latency. Our study showcases the feasibility and benefits of self-adaptive mechanisms in blockchain systems through literature review and experimentation, pointing towards a future where blockchain infrastructure can dynamically adjust to achieve optimal operation without manual intervention.

Combining all the insights from our work, a detailed picture emerges on the necessity and methodology for enhancing the efficiency and robustness of blockchain systems. We highlight the complex interplay between system configurations, workload characteristics, and optimization strategies, advocating for a balanced approach that considers all aspects of blockchain operation. The research collectively advances the field by not only diagnosing existing limitations but also providing validated solutions that pave the way for more adaptive, efficient, and user-friendly blockchain ecosystems.

There are various future research directions that can extend and enrich our current work in several ways. As the concept of self-driving blockchains advances, ensuring the security and integrity of these autonomous systems becomes paramount. Future work could focus on developing robust security frameworks that can dynamically adapt to new threats, ensuring the safety and reliability of decentralized systems. Additionally, there are growing concerns over the environmental impact of blockchain technologies, so future research could aim to dynamically optimize the energy consumption and carbon footprint of blockchain operations.

Bibliography

[1] Orlenys Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. "Caterpillar: A business process execution engine on the Ethereum blockchain." In: *Software: Practice and Experience* (May 2019). DOI: 10.1002/spe.2702.

[2] Vitalik Buterin et al. "A next-generation smart contract and decentralized application platform." In: *white paper* 3.37 (2014), pp. 2–1.

[3] Ashar Ahmad, Muhammad Saad, Joongheon Kim, DaeHun Nyang, and David Mohaisen. "Performance Evaluation of Consensus Protocols in Blockchain-based Audit Systems." In: *2021 International Conference on Information Networking (ICOIN)*. 2021, pp. 654–656. DOI: 10.1109/ICOIN50884.2021.9333867.

[4] Xiaoqiang Ding, Liushun Zhao, Lailong Luo, et al. "Gauze: enabling communication-friendly block synchronization with cuckoo filter." In: *Frontiers of Computer Science* 17.3 (2023), p. 173403.

[5] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. "Permissioned Blockchains: Properties, Techniques and Applications." In: New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450383431. URL: https://doi.org/10.1145/3448016.3457539.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, et al. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains." In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. Porto, Portugal: ACM, 2018, 30:1–30:15. ISBN: 978-1-4503-5584-1. DOI: 10.1145/3190508.3190538. URL: http://doi.acm.org/10.1145/3190508.3190538.

[7] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. "Corda: an introduction." In: *R3 CEV, August* 1 (2016), p. 15.

[8] Gideon Greenspan et al. "Multichain private blockchain-white paper." In: *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf* (2015), pp. 57–60.

[9] JP Morgan. "Quorum whitepaper." In: *New York: JP Morgan Chase* (2016).

[10] Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thajchayapong. "Performance Analysis of Private Blockchain Platforms in Varying Workloads." In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 2017, pp. 1–6. DOI: 10.1109/ICCCN.2017.8038517.

[11] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, et al. “Blockchains vs. Distributed Databases: Dichotomy and Fusion.” In: *Proceedings of the 2021 International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450383431. URL: https://doi.org/10.1145/3448016.3452789.

[12] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. “A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities.” In: *IEEE Access* 7 (2019), pp. 117134–117151. DOI: 10.1109/ACCESS.2019.2936094.

[13] Ye Guo and Chen Liang. “Blockchain application and outlook in the banking industry.” In: *Financial innovation* 2 (2016), pp. 1–12.

[14] https://dappradar.com/rankings/protocol/ethereum. [Online; accessed 12-March-2024]. 2024.

[15] https://www.hyperledger.org/learn/use-case-tracker. [Online; accessed 12-March-2024]. 2024.

[16] Sara Bergman, Mikael Asplund, and Simin Nadjm-Tehrani. “Permissioned blockchains and distributed databases: A performance study.” In: *Concurrency and Computation: Practice and Experience* 32.12 (2020). e5227 cpe.5227, e5227. DOI: https://doi.org/10.1002/cpe.5227. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5227. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5227.

[17] Mohammad Jabed Morshed Chowdhury, Alan Colman, Muhammad Ashad Kabir, Jun Han, and Paul Sarda. “Blockchain Versus Database: A Critical Analysis.” In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2018, pp. 1348–1353. DOI: 10.1109/TrustCom/BigDataSE.2018.00186.

[18] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, et al. “Untangling Blockchain: A Data Processing View of Blockchain Systems.” In: *IEEE Transactions on Knowledge and Data Engineering* 30.7 (2018), pp. 1366–1385. DOI: 10.1109/TKDE.2017.2781227.

[19] Olivier Rikken, Marijn Janssen, and Zenlin Kwee. “Governance challenges of blockchain and decentralized autonomous organizations.” In: *Information Polity* 24 (Nov. 2019), pp. 1–21. DOI: 10.3233/IP-190154.

[20] Mayank Raikwar, Danilo Gligoroski, and Goran Velinov. “Trends in Development of Databases and Blockchain.” In: *2020 Seventh International Conference on Software Defined Systems (SDS)*. 2020, pp. 177–182. DOI: 10.1109/SDS49854.2020.9143893.

[21] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. “A review on consensus algorithm of blockchain.” In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2017, pp. 2567–2572. DOI: 10.1109/SMC.2017.8123011.

[22] Aravind Ramachandran and Dr. Murat Kantarcioglu. “Using Blockchain and smart contracts for secure data provenance management.” In: *arXiv* (2017).

[23] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. "An Overview of Smart Contract and Use Cases in Blockchain Technology." In: *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2018, pp. 1–4. DOI: 10.1109/ICCCNT.2018.8494045.

[24] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. "Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric." In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 221–234. ISBN: 9781450383431. DOI: 10.1145/3448016.3452823. URL: https://doi.org/10.1145/3448016.3452823.

[25] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. "Blurring the Lines Between Blockchains and Database Systems: The Case of Hyperledger Fabric." In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Amsterdam, Netherlands: ACM, 2019. ISBN: 978-1-4503-5643-5. DOI: 10.1145/3299869.3319883. URL: http://doi.acm.org/10.1145/3299869.3319883.

[26] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, et al. "A Transactional Perspective on Execute-Order-Validate Blockchains." In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD '20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 543–557. ISBN: 9781450367356. DOI: 10.1145/3318464.3389693. URL: https://doi.org/10.1145/3318464.3389693.

[27] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. "How To Optimize My Blockchain? A Multi-Level Recommendation Approach." In: *Proc. ACM Manag. Data* 1.1 (May 2023). DOI: 10.1145/3588704. URL: https://doi.org/10.1145/3588704.

[28] Jeeta Ann Chacko, Nino Richter, Ruben Mayer, and Hans-Arno Jacobsen. "Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric." In: *Proceedings of the 24th International Middleware Conference: Demos, Posters and Doctoral Symposium*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 31–32. ISBN: 9798400704291. URL: https://doi.org/10.1145/3626564.3629098.

[29] Akash Takyar. 2023. URL: https://www.leewayhertz.com/cost-of-blockchain-implementation.

[30] Sudeep Srivastava. 2023. URL: https://appinventiv.com/guide/blockchain-app-development-cost/.

[31] Anurag Jain. 2023. URL: https://oyelabs.com/blockchain-app-development-cost/.

[32] 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)*. 2018, pp. 264–276. DOI: 10.1109/MASCOTS.2018.00034.

[33] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. "FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second." In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2019, pp. 455–463. DOI: 10.1109/BLOC.2019.8751452.

[34] Zsolt István, Alessandro Sorniotti, and Marko Vukolić. “Streamchain: Do blockchains need blocks?” In: *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. 2018, pp. 1–6.

[35] Shashank Motepalli and Hans-Arno Jacobsen. *Analyzing Geospatial Distribution in Blockchains*. 2023. arXiv: 2305.17771 [cs.DC].

[36] Performance considerations. 2023. URL: https://hyperledger-fabric.readthedocs.io/en/latest/performance.html.

[37] Michel Rauchs, Apolline Blandin, Keith Bear, and Stephen B McKeon. “2nd global enterprise blockchain benchmarking study.” In: *Available at SSRN 3461765* (2019).

[38] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. *Should my Blockchain Learn to Drive? A Study of Hyperledger Fabric*. 2024. arXiv: 2406.06318.

[39] Jeeta Ann Chacko, Ruben Mayer, Alan Fekete, Vincent Gramoli, and Hans-Arno Jacobsen. “A Comprehensive Study on Benchmarking Permissioned Blockchains.” In: *Performance Evaluation and Benchmarking*. Ed. by Raghunath Nambiar and Meikel Poess. Cham: Springer Nature Switzerland, 2024.

[40] https://hyperledger.github.io/caliper/. [Online; accessed 20-June-2024]. 2020.

[41] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. “Medrec: Using blockchain for medical data access and permission management.” In: *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE. 2016, pp. 25–30.

[42] Tomas Mikula and Rune Hylsberg Jacobsen. “Identity and access management with blockchain in electronic healthcare records.” In: *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE. 2018, pp. 699–706.

[43] Emre Yavuz, Ali Kaan Koç, Umut Can Çabuk, and Gökhan Dalkılıç. “Towards secure e-voting using ethereum blockchain.” In: *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE. 2018, pp. 1–7.

[44] Guido Perboli, Stefano Musso, and Mariangela Rosano. “Blockchain in logistics and supply chain: A lean approach for designing real-world use cases.” In: *IEEE Access* 6 (2018), pp. 62018–62028.

[45] Arati Baliga, Nitesh Solanki, Shubham Verekar, et al. “Performance Characterization of Hyperledger Fabric.” In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, pp. 65–74. DOI: 10.1109/CVCBT.2018.00013.

[46] Michael Stonebraker and Ugur Çetintemel. “"One Size Fits All": An Idea Whose Time Has Come and Gone.” In: *Proceedings of the 21st International Conference on Data Engineering*. ICDE ’05. USA: IEEE Computer Society, 2005, pp. 2–11. ISBN: 0769522858. DOI: 10.1109/ICDE.2005.1. URL: https://doi.org/10.1109/ICDE.2005.1.

[47] Robin Rehrmann, Carsten Binnig, Alexander Böhm, et al. “OLTPshare: The Case for Sharing in OLTP Workloads.” In: *Proc. VLDB Endow.* 11.12 (2018). ISSN: 2150-8097. DOI: 10.14778/3229863.3229866. URL: https://doi.org/10.14778/3229863.3229866.

[48] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. "FabricCRDT: A Conflict-Free Replicated Datatypes Approach to Permissioned Blockchains." In: *Proceedings of the 20th International Middleware Conference.* Middleware '19. Davis, CA, USA: Association for Computing Machinery, 2019, pp. 110—122. ISBN: 9781450370097. DOI: 10.1145/3361525.3361540. URL: https://doi.org/10.1145/3361525.3361540.

[49] Parinaz Ameri. "On a self-tuning index recommendation approach for databases." In: *2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW).* 2016, pp. 201–205. DOI: 10.1109/ICDEW.2016.7495648.

[50] Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. "Query Recommendations for Interactive Database Exploration." In: *Scientific and Statistical Database Management.* Ed. by Marianne Winslett. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 3–18. ISBN: 978-3-642-02279-1.

[51] Sadhana J. Kamatkar, Ajit Kamble, Amelec Viloria, Lissette Hernández-Fernandez, and Ernesto García Cali. "Database Performance Tuning and Query Optimization." In: *Data Mining and Big Data.* Ed. by Ying Tan, Yuhui Shi, and Qirong Tang. Cham: Springer International Publishing, 2018, pp. 3–11. ISBN: 978-3-319-93803-5.

[52] Christopher Klinkmüller, Alexander Ponomarev, An Binh Tran, Ingo Weber, and Wil van der Aalst. "Mining blockchain processes: Extracting process mining data from blockchain applications." In: *International Conference on Business Process Management.* Springer. 2019, pp. 71–86.

[53] Roman Mühlberger, Stefan Bachhofner, Claudio Di Ciccio, Luciano Garcıa-Bañuelos, and Orlenys López-Pintado. "Extracting event logs for process mining from data stored on the blockchain." In: *International Conference on Business Process Management.* Springer. 2019, pp. 690–703.

[54] Stefan Tönnissen and Frank Teuteberg. "Using Blockchain Technology for Cross-Organizational Process Mining–Concept and Case Study." In: *International Conference on Business Information Systems.* Springer. 2019, pp. 121–131.

[55] Frank Duchmann and Agnes Koschmider. "Validation of smart contracts using process mining." In: *ZEUS. CEUR workshop proceedings.* Vol. 2339. 2019, pp. 13–16.

[56] Richard Hobeck, Christopher Klinkmüller, Hmn Dilum Bandara, Ingo Weber, and Wil Van der Aalst. *Process Mining on Blockchain Data: a Case Study of Augur.* Tech. rep. EasyChair, 2021.

[57] Surajit Chaudhuri and Vivek Narasayya. "Self-Tuning Database Systems: A Decade of Progress." In: *Proceedings of the 33rd International Conference on Very Large Data Bases.* VLDB '07. Vienna, Austria: VLDB Endowment, 2007, pp. 3–14. ISBN: 9781595936493.

[58] Ji Zhang, Yu Liu, Ke Zhou, et al. "An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning." In: *Proceedings of the 2019 International Conference on Management of Data.* SIGMOD '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 415–432. ISBN: 9781450356435. DOI: 10.1145/3299869.3300085. URL: https://doi.org/10.1145/3299869.3300085.

[59] Sushil Kumar. "Oracle database 10g: The self-managing database." In: *White Paper* (2003).

[60] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. "ElasTraS: An Elastic, Scalable, and Self-Managing Transactional Database for the Cloud." In: 38.1 (2013). ISSN: 0362-5915. DOI: 10.1145/2445583.2445588. URL: https://doi.org/10.1145/2445583.2445588.

[61] Guilain Leduc, Sylvain Kubler, and Jean-Philippe Georges. "Sabine: Self-Adaptive BlockchaIn coNsEnsus." In: *2022 9th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2022, pp. 234–240. DOI: 10.1109/FiCloud57274.2022.00039.

[62] Na Ruan, Dongli Zhou, and Weijia Jia. "Ursa: Robust Performance for Nakamoto Consensus with Self-Adaptive Throughput." In: *ACM Trans. Internet Technol.* 20.4 (2020). ISSN: 1533-5399. DOI: 10.1145/3412341. URL: https://doi.org/10.1145/3412341.

[63] Chenyuan Wu, Bhavana Mehta, Mohammad Javad Amiri, Ryan Marcus, and Boon Thau Loo. "AdaChain: A Learned Adaptive Blockchain." In: *Proc. VLDB Endow.* 16.8 (2023), pp. 2033–2046. ISSN: 2150-8097. DOI: 10.14778/3594512.3594531. URL: https://doi.org/10.14778/3594512.3594531.

[64] Mingxuan Li, Yazhe Wang, Shuai Ma, et al. "Auto-Tuning with Reinforcement Learning for Permissioned Blockchain Systems." In: *Proc. VLDB Endow.* 16.5 (2023), pp. 1000–1012. ISSN: 2150-8097. DOI: 10.14778/3579075.3579076. URL: https://doi.org/10.14778/3579075.3579076.

[65] Mohammadreza Rasolroveicy. "A Self-Adaptive Blockchain Framework to Balance Performance, Security, and Energy Consumption in IoT applications." In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. 2020, pp. 243–245. DOI: 10.1109/ACSOS-C51401.2020.00068.

[66] Jie Xu, Qingyuan Xie, Sen Peng, Cong Wang, and Xiaohua Jia. "AdaptChain: Adaptive Scaling Blockchain With Transaction Deduplication." In: *IEEE Transactions on Parallel and Distributed Systems* 34.6 (2023), pp. 1909–1922. DOI: 10.1109/TPDS.2023.3267071.

[67] Frank Christian Geyer, Hans-Arno Jacobsen, Ruben Mayer, and Peter Mandl. "An End-to-End Performance Comparison of Seven Permissioned Blockchain Systems." In: *Proceedings of the 24th International Middleware Conference*. Middleware '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 71–84. DOI: 10.1145/3590140.3629106. URL: https://doi.org/10.1145/3590140.3629106.

[68] Caixiang Fan, Sara Ghaemi, Hamzeh Khazaei, and Petr Musilek. "Performance Evaluation of Blockchain Systems: A Systematic Survey." In: *IEEE Access* 8 (2020), pp. 126927–126950. DOI: 10.1109/ACCESS.2020.3006078.

Appendix A

Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric

Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric

Jeeta Ann Chacko
chacko@in.tum.de
Technical University of Munich

Ruben Mayer
mayerr@in.tum.de
Technical University of Munich

Hans-Arno Jacobsen
jacobsen@eecg.toronto.edu
University of Toronto

Abstract

Permissioned blockchain systems promise to provide both decentralized trust and privacy. Hyperledger Fabric is currently one of the most wide-spread permissioned blockchain systems and is heavily promoted both in industry and academia. Due to its optimistic concurrency model, the transaction failure rates in Fabric can become a bottleneck. While there is active research to reduce failures, there is a lack of understanding on their root cause and, consequently, a lack of guidelines on how to configure Fabric optimally for different scenarios. To close this gap, in this paper, we first introduce a formal definition of the different types of transaction failures in Fabric. Then, we develop a comprehensive testbed and benchmarking system, HyperLedgerLab, along with four different chaincodes that represent realistic use cases and a chaincode/workload generator. Using HyperLedgerLab, we conduct exhaustive experiments to analyze the impact of different parameters of Fabric such as block size, endorsement policies, and others, on transaction failures. We further analyze three recently proposed optimizations from the literature, Fabric++, Streamchain and FabricSharp, and evaluate under which conditions they reduce the failure rates. Finally, based on our results, we provide recommendations for Fabric practitioners on how to configure the system and also propose new research directions.

CCS Concepts

• **Information systems → Data management systems**.

Keywords

Blockchains, transaction failures, concurrency

ACM Reference Format:
Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 18–27, 2021, Virtual Event , China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3448016.3452823

1 Introduction

With the advent of Bitcoin [30], a renewed interest in decentralized trust and immutable records emerged. But enterprises are wary to adopt a framework which allows open participation and offers limited throughput. This gave birth to the concept of permissioned blockchains which provide higher transaction rates and decentralized trust [5, 18, 39]. As there exists a trade-off between decentralization, consistency and scalability in blockchains [54], such permissioned blockchains need to restrict access to a set of authorized participants. Permissioned blockchains are gaining increasing popularity since enterprises can now identify use cases which can be implemented more efficiently on blockchains. Following this trend, many permissioned blockchains such as Corda [5], Multichain [18] and Quorum [39] appeared. Currently, Hyperledger Fabric (a.k.a. Fabric) [1] is a widely used permissioned blockchain framework. For instance, a recent survey by Rauchs et al. shows that 48% of all live permissioned blockchain projects in the Cambridge Centre for Alternative Finance dataset build on Fabric [41, 49].

Despite its wide adoption, Fabric still has its limitations. It follows an optimistic concurrency control model which causes transaction failures when conflicting transactions are concurrently executed [19]. In our experiments, we observed that more than 40% of the transactions failed due to concurrency-related conflicts in a realistic scenario (Electronic Health Record, cf. Table 2). There is active research to improve the throughput and reduce transaction failures in Fabric [15–17, 24, 43, 44, 47]. However, there is no formal definition for transaction failures, and no comprehensive study on their cause and the parameters that influence them. As a result, research on Fabric transaction failures falls short in exploring the full range of the problem and trade-offs that are involved. For example, Fabric++ by Sharma et al. [44] reorders transactions to reduce a specific type of failures (Multi-Version Concurrency Control (MVCC) read conflicts), but neglects the effect on other failure types. Furthermore, different papers evaluate their approach to reduce concurrency-related transaction failures using distinct smart contracts and workloads, which hinders a direct and fair comparison [15–17, 24, 43, 44, 47]. For example, Sharma et al. [44] use a smart contract based on an asset transfer scenario while István et al. [24] use Fabcoin [1], a digital currency inspired by Bitcoin [30].

In this paper, we close this gap by performing an extensive study on transaction failures in Fabric using a comprehensive benchmarking system. We provide the following contributions:

(1) We formally define the different types of concurrency-related transaction failures in order to build a solid foundation for our study and for further research.
(2) We extensively study the various parameters influencing different transaction failures in Fabric as well as for three recent optimization techniques, Fabric++ [44], Streamchain [24] and FabricSharp [43]. Our study reveals surprising insights and trade-offs regarding the configuration of Fabric. For instance, the *block size* has a significant impact on the number

SIGMOD '21, June 18–27, 2021, Virtual Event , China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452823

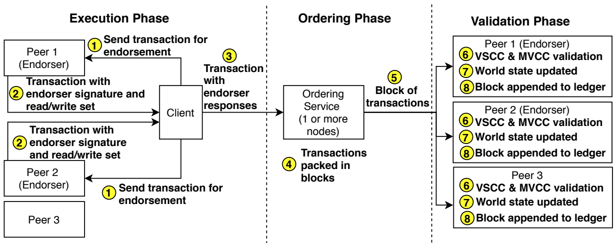

Execution Phase
Ordering Phase
Validation Phase
Peer 1 (Endorser)
1 Send transaction for endorsement
3 Transaction with endorser responses
2 Transaction with endorser signature and read/write set
Client
2 Transaction with endorser signature and read/write set
Peer 2 (Endorser)
1 Send transaction for endorsement
Peer 3
Ordering Service (1 or more nodes)
4 Transactions packed in blocks
5 Block of transactions
Peer 1 (Endorser)
6 VSCC & MVCC validation
7 World state updated
8 Block appended to ledger
Peer 2 (Endorser)
6 VSCC & MVCC validation
7 World state updated
8 Block appended to ledger
Peer 3
6 VSCC & MVCC validation
7 World state updated
8 Block appended to ledger

Figure 1: Transaction Flow in Hyperledger Fabric

of transaction failures at various transaction arrival rates—with the right block size, transaction failures could be reduced by up to 60%.

(3) For realistic evaluations and controlled experiments, we develop a new testbed integrated with an extension of the Caliper benchmarking system, HyperLedgerLab [23], which includes four new smart contracts that represent realistic use cases as well as a chaincode and workload generator. We released HyperLedgerLab as well as all the chaincodes and the generator as an open source project, so that other researchers can benefit from it and compare their work in a fair way.

(4) We identify best practices and principles for Fabric developers and also derive promising research directions that the scientific community can pursue in the future.

The rest of this paper is organized as follows. In Section 2, we provide the technical background on Fabric. We introduce and formalize the different transaction failures in Section 3. In Section 4, we explain our new benchmarking system HyperLedgerLab, before we describe and discuss our experimental results in Section 5. Finally, in Section 6, we summarize the insights and lessons learned, discuss related work in Section 7 and conclude the paper in Section 8.

2 Hyperledger Fabric

Fabric is a popular open source permissioned blockchain system established under the Linux foundation [1]. It is the first blockchain system that supports the creation of smart contracts in general purpose languages. Fabric allows clients to submit *transactions* to a blockchain system which offers decentralized control of a shared, distributed state, i.e., there is not a single trusted entity that decides about the current state of the system. All possible functions that can be invoked by a transaction are defined in a *smart contract*, which is called *chaincode* in Fabric jargon. The distributed state, called *world state*, is maintained as a versioned key-value store—Fabric currently supports LevelDB [11] and CouchDB [2]. Each key has a version number which is updated with every write. The *distributed ledger* maintains the complete history of all the transactions (successful and failed) in the network which are grouped into blocks. Both the world state and the distributed ledger are replicated on a set of distributed nodes that are registered on the Fabric network—called *peers*. Peers receive blocks of transactions from an *ordering service* that guarantees the ordered delivery; more precisely, all peers receive all transactions in the same order. They validate every transaction independently and update their copy of the world state and the ledger accordingly. For fault tolerance, the ordering service can be replicated and uses a consensus protocol (e.g., Paxos [25] or Raft [32]) to reach an agreement about the order of all transactions. The ordering service handles multiple *channels*, i.e., private communication routes between different Fabric components. *Endorsers* are a subset of the peers that have the additional role of endorsing transactions submitted by the clients, i.e., they simulate the execution of the transaction to generate *read sets* and *write sets* based on the current world state. An *endorsement policy* defines the number of endorsements that are required for a transaction to be accepted as valid. Finally, peers are grouped into *organizations* which typically correspond to real organizations or branches of an enterprise; these organizations can play an important role in the endorsement policy.

The transaction flow in Fabric follows three phases: execution, ordering and validation. This is referred to as the Execute-Order-Validate (E-O-V) model and is visualized in Figure 1. Each phase and the different steps shown in Figure 1 are described below.

Execution Phase

Step 1: The client sends a transaction to all the endorsers. The transaction can include multiple reads and writes to one or more keys in the world state.

Step 2: The endorsers simulate the execution of the transaction on the world state and generate a read/write set that corresponds to the current world state of every key in the transaction. Then, the endorsers send a response back to the client that contains their own signature and the read/write set. This distributed execution of transactions on the endorsers helps to maintain trust without a centralized authority.

Step 3: The client collects the endorsing peers responses and sends them to the ordering service nodes. Optionally, the client may check the validity of the endorsing peers signatures and the consistency between the read/write set received from different peers. These are mandatorily checked later in the validation phase. Doing this check, the client can help detect transaction failures early in the transaction flow to reduce overhead.

Ordering Phase

Step 4: The ordering service orders the transactions received from the client using a consensus protocol [25, 32]. A transaction block is created based on three conditions: if a fixed duration of time has elapsed (*block timeout*), if a fixed number of transactions have been received (*block size*) or if the total size of transactions has reached a fixed limit (*block max bytes*).

Step 5: The block of transactions is then sent to all the peers.

Validation Phase

Step 6: Every peer, upon receiving a block of transactions from the ordering service, validates each transaction in the block independently. A peer checks if a sufficient number of valid endorsing peer signatures, based on the endorsement policy, have been collected (Validation System Chaincode (VSCC) validation). Then, the peer verifies if the version of every key in the read set of each transaction is equal to the version of the same key in the current world state (MVCC validation).

Step 7: If VSCC and MVCC validation checks pass, the write sets of the transactions are applied to the world state. If any of the validation checks fail, the client is notified that the transaction aborted, and the world state does not change.

P: endorsing peers	$\mathcal{V}^R$: versions of keys in a read set
T: transactions	$\mathfrak{B}^W$: values of keys in a write set
B: blocks of transactions	$\mathcal{K}^X$: keys in the world state
$\mathcal{K}^R$: keys in a read set	$\mathcal{V}^X$: versions of keys in the world state
$\mathcal{K}^W$: keys in a write set	$\mathfrak{B}^X$: values of keys in the world state

Table 1: Notations for sets

Step 8: The validated block containing both aborted and committed transactions is appended to the ledger. The commit or abort status of every transaction is logged.

3 Types of Transaction Failures

The endorsement phase and the other two phases (ordering and validation) happen in parallel. Hence, Fabric may execute transactions in the endorsement phase before previous transactions are ordered and committed. Thus, transactions are not always executed on the latest state of the world state in the endorsement phase. We have identified three types of transaction failures which are caused by this problem. Before formalizing them, we define basic concepts.

3.1 Basic Concepts

The following definitions use the notations shown in Table 1.

Definition 1: Read Set. The read set $RS_{T_i P_a}$ of a transaction $T_i \in T$ generated by an endorsing peer $P_a \in P$ is represented by a set of $m \in \mathbb{N}_0$ ordered pairs of keys $\mathcal{K}_k^R \in \mathcal{K}^R$ and corresponding versions $\mathcal{V}_k^R \in \mathcal{V}^R$:

$$RS_{T_i P_a} = \{(\mathcal{K}_1^R, \mathcal{V}_1^R), \ldots, (\mathcal{K}_m^R, \mathcal{V}_m^R)\}$$

Definition 2: Write Set. The write set $WS_{T_i P_a}$ of a transaction $T_i \in T$ generated by an endorsing peer $P_a \in P$ is represented by a set of $n \in \mathbb{N}_0$ ordered pairs of keys $\mathcal{K}_k^W \in \mathcal{K}^W$ and corresponding values $\mathfrak{B}_k^W \in \mathfrak{B}^W$:

$$WS_{T_i P_a} = \{(\mathcal{K}_1^W, \mathfrak{B}_1^W), \ldots, (\mathcal{K}_n^W, \mathfrak{B}_n^W)\}$$

Definition 3: World State. The world state WX is represented by an ordered set of $o \in \mathbb{N}_0$ keys $\mathcal{K}_k^X \in \mathcal{K}^X$, corresponding versions $\mathcal{V}_k^X \in \mathcal{V}^X$ and values $\mathfrak{B}_k^X \in \mathfrak{B}^X$:

$$WX = \{(\mathcal{K}_1^X, \mathcal{V}_1^X, \mathfrak{B}_1^X), \ldots, (\mathcal{K}_o^X, \mathcal{V}_o^X, \mathfrak{B}_o^X)\}$$

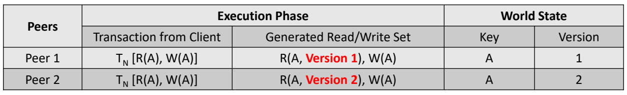
Definition 4: Transaction Dependency. A transaction T_i is dependent on transaction T_j if the read set $RS_{T_i P_a}$ of T_i contains a key which is also present in the write set $WS_{T_j P_a}$ of T_j:

$$RS_{T_i P_a} \cap WS_{T_j P_a} \neq \emptyset \textit{ and } i \neq j$$

3.2 Transaction Failures

3.2.1 Endorsement policy failures: All transactions need to be endorsed by the endorsing peers in the execution phase (cf. Section 2). The endorsement of transactions can fail for multiple reasons, such as invalid endorser signatures or other technical reasons. Most of the possible causes for *endorsement policy failures* are due to misconfigurations and unrelated to concurrency. In this study, we only consider endorsement policy failures caused by a read/write set mismatch, as described in the following.

The key-value store, which maintains the world state, is updated by each peer independently in the validation phase. Therefore, transient world state inconsistencies between the peers are possible. At the same time, the endorsing peers use the world state to generate read/write sets in the execution phase. Thus, the world state inconsistencies lead to a read/write set mismatch in the endorsement response causing an endorsement policy failure of the transaction.



Peers	Execution Phase		World State	
	Transaction from Client	Generated Read/Write Set	Key	Version
Peer 1	T_N [R(A), W(A)]	R(A, Version 1), W(A)	A	1
Peer 2	T_N [R(A), W(A)]	R(A, Version 2), W(A)	A	2

Figure 2: Example of an endorsement policy failure

An example is illustrated in Figure 2. Here, the world states of Peer 1 and Peer 2 are inconsistent, i.e., the version of Key A is different on both peers. At this point in time a new transaction T_N which reads Key A is being executed on both Peer 1 and Peer 2. This execution would give rise to different results on both peers since the version of Key A is different. This results in an endorsement policy failure.

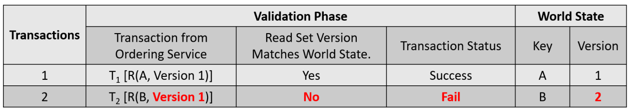
Formally, an endorsement policy failure occurs when there exist two different endorsing peers $P_a, P_b \in P$ that both endorse the same transaction $T_i \in T$, such that there is the same read key $\mathcal{K}_k^R \in \mathcal{K}^R$ in the corresponding read sets $RS_{T_i P_a}$ and $RS_{T_i P_b}$, but the versions $\mathcal{V}_k^R$ of that key in $RS_{T_i P_a}$ and $RS_{T_i P_b}$ are different:

$$\begin{gathered} \exists\, T_i \in T \;\wedge\; P_a, P_b \in P \;\wedge\; \mathcal{K}_k^R \in \mathcal{K}^R \;\wedge\; \mathcal{V}_k^R \in \mathcal{V}^R \\ \wedge\; a \neq b \textit{ such that } \mathcal{K}_k^R \textit{ of } RS_{T_i P_a} == \mathcal{K}_k^R \textit{ of } RS_{T_i P_b} \;\wedge \\ \mathcal{V}_k^R \textit{ of } RS_{T_i P_a} \neq \mathcal{V}_k^R \textit{ of } RS_{T_i P_b} \end{gathered} \quad (1)$$

3.2.2 MVCC read conflicts: *MVCC read conflicts* are a well known problem in systems that have a multi-version view of each key [29]. Every successful write on a key will increment the version and update the value. Every successful read will get the current version of the key in the world state along with its value. While a transaction moves from the execution phase to the validation phase, other transactions can get validated and committed, thereby updating the world state. Therefore, when a transaction reaches the validation stage, the world state may have changed and be different from the endorsement.

Transactions that access the same key and thereby create a dependency are subject to *MVCC read conflicts*. We further distinguish between *inter-block MVCC read conflicts* and *intra-block MVCC read conflicts*. Transaction failures caused due to a dependency among transactions in the same block are defined as intra-block MVCC read conflicts. If the cause is a dependency among transactions in different blocks, we call this inter-block MVCC read conflicts.

Though the basic cause for both conflicts is the transaction dependency, other parameters such as block size influence these failures differently. For example, if two dependent transactions are submitted far apart in time and if they are included in different blocks, the first block may get committed before the second transaction is endorsed. Thus, both transactions could potentially succeed. However, if in the same scenario, the block size was very large, they could get endorsed and then ordered into the same block. This would cause one of the transactions to fail validation. A further difference is that intra-block MVCC read conflicts can potentially be resolved by transaction reordering [44] while inter-block MVCC read conflicts cannot be resolved in such a way.



Transactions	Validation Phase			World State	
	Transaction from Ordering Service	Read Set Version Matches World State.	Transaction Status	Key	Version
1	T_1 [R(A, Version 1)]	Yes	Success	A	1
2	T_2 [R(B, Version 1)]	No	Fail	B	2

Figure 3: Example of an MVCC read conflict

An example of MVCC read conflicts is illustrated in Figure 3. Here, Transaction 1 (T_1) reads Key A whose version in the world state is the same as the version in the read set of the transaction. Hence, the read set contains the latest value of Key A. Whereas for Transaction 2 (T_2) that reads Key B, there are different versions in the world state and the read set. This implies that T_2 is accessing an older version of the key and therefore fails.

Formally, an MVCC read conflict occurs for a transaction when there exists a key $\mathcal{K}_k^R \in \mathcal{K}^R$ in the read set that matches a key $\mathcal{K}_k^X \in \mathcal{K}^X$ in the world state, but the version $\mathcal{V}_k^R \in \mathcal{V}^R$ of that key in the read set does not match the corresponding version $\mathcal{V}_k^X \in \mathcal{V}^X$ in the world state:

$$\begin{aligned}\exists\, \mathcal{K}_k^R \in \mathcal{K}^R \wedge \mathcal{K}_k^X \in \mathcal{K}^X \wedge \mathcal{V}_k^R \in \mathcal{V}^R \wedge \mathcal{V}_k^X \in \mathcal{V}^X \\ \textit{such that}\, \mathcal{K}_k^R == \mathcal{K}_k^X \wedge \mathcal{V}_k^R \neq \mathcal{V}_k^X\end{aligned} \tag{2}$$

An MVCC read conflict is an intra-block conflict when Equation 2 holds and additionally, the cause of the MVCC read conflict is due to the dependency between two transactions T_i and T_j in the same block $b_l \in B$, i.e., the read set of T_i contains a key $\mathcal{K}_k^R$ that is also present in the write set of T_j, but T_j occurs before T_i in the block:

$$\begin{aligned}\exists\, \mathcal{K}_k^R \in \mathcal{K}^R \wedge \mathcal{K}_k^W \in \mathcal{K}^W \wedge T_i, T_j \in T \\ \wedge\, T_i, T_j \in b_l \wedge b_l \in B \wedge j < i \\ \textit{such that}\, \mathcal{K}_k^R \textit{ of } T_i == \mathcal{K}_k^W \textit{ of } T_j\end{aligned} \tag{3}$$

Similarly, an MVCC read conflict is an inter-block conflict when Equation 2 holds and additionally, the MVCC read conflict is due to two transactions T_i and T_j in two different blocks b_l and b_m, where the transaction T_j that writes the key occurs in an earlier block than the transaction T_i that reads the key:

$$\begin{aligned}\exists\, \mathcal{K}_k^R \in \mathcal{K}^R \wedge \mathcal{K}_k^W \in \mathcal{K}^W \wedge T_i, T_j \in T \wedge T_i \in b_l \\ \wedge\, T_j \in b_m \wedge b_l, b_m \in B \wedge m < l \\ \textit{such that}\, \mathcal{K}_k^R \textit{ of } T_i == \mathcal{K}_k^W \textit{ of } T_j\end{aligned} \tag{4}$$

3.2.3 Phantom read conflicts: Fabric supports *range queries* which read a set of keys from the key-value store. Given a start and end key, range queries read all the keys from the world state within this interval. In the validation phase, the entire range of keys is checked to ensure that no keys were inserted, deleted or updated within that range. If at least one key in the range has been inserted, deleted or updated, the transaction is aborted, and such failures are identified as *phantom read conflicts*.

Formally, a phantom read conflict occurs when Equation 2 holds or under one of the following conditions. First, when a key $\mathcal{K}_k^R \in \mathcal{K}^R$ in the read set is not present in the world state. Second, when there exists a key $\mathcal{K}_k^X \in \mathcal{K}^X$ in the world state that is within the key interval $[i, j]$ of the range query, but is not present in the read set:

$$\exists\, \mathcal{K}_k^R \notin \mathcal{K}^X \vee \mathcal{K}_k^X \notin \mathcal{K}^R \textit{ where } i \leq k \leq j \tag{5}$$

Phantom read conflicts are essentially MVCC read conflicts since the cause for both is the dependency between transactions. However, Fabric identifies them separately since they are specific to range queries. The separate analysis of phantom read conflicts is justified, as we expect transactions with range queries to have more transaction dependencies than other transactions. For example, a range query that reads all the keys in the world state is dependent on every other write transaction.

4 Experimental Methodology

4.1 HyperLedgerLab

We developed HyperLedgerLab [23] as an automated solution for setting up the experimental environment. It initially uses OpenStack [6] APIs to commission the required infrastructure. Then, a Kubernetes [37] cluster is launched on this infrastructure. Next, the Fabric network is deployed on the cluster and the benchmark is started. The Fabric network can be deleted and recreated with different configurations to run the benchmark multiple times. The two benchmarks currently available for permissioned blockchains are Blockbench [12] and Hyperledger Caliper [20]. During the inception of this paper, Blockbench only supported Fabric version 0.6[1]. Therefore, we chose to integrate Caliper with HyperLedgerLab. It has a Fabric adapter that uses the nodeJS-SDK to communicate with the Fabric network via gRPC. Caliper monitors the performance of the blockchain and gathers metrics. We extended Caliper to collect all the different types of transaction failures necessary for our evaluation. We then developed four use-case based chaincodes and workloads which are explained in Section 4.3 and a chaincode and workload generator which is explained in Section 4.4.

4.2 Cluster Setup

We used a Kubernetes cluster consisting of one command line interface (CLI) node, three controller nodes, one load balancer, one network file system (NFS) node and multiple worker nodes. Every node runs on a Ubuntu Xenial (16.04) virtual machine. The CLI node and worker nodes have 16 vCPUs and 41 GB RAM each, the controller nodes have four vCPUs and 20 GB RAM each, and the load balancer and NFS node have two vCPUs and 10 GB RAM each. The workers host the pods for the different Fabric components such as peers and orderers. The controller nodes of the Kubernetes cluster are responsible for scheduling. The peers and orderers are deployed on the worker nodes in a round-robin fashion by the Kubernetes scheduler.

We use two different cluster setups for our experiments. The first setup (*C1*) uses 3 worker nodes on which 4 peers and 3 orderers are launched and 5 client processes which are launched on the CLI node. The second setup (*C2*) uses 32 worker nodes on which 32 peers and 3 orderers are launched. Also, in this setup 25 client processes are launched on the CLI node. Fabric 1.4 supports both the Solo and Kafka ordering service. We use Kafka for our experiments since Solo is not used in production.

4.3 Use-Case Based Chaincodes & Workloads

We developed four different chaincodes based on popular use cases from various disciplines to diversify our evaluation and make it realistic. The functions of each chaincode are shown in Table 2 along with the number and type of read and write operations performed by each function. All the read and write functions access keys randomly. The keys accessed by the range reads are described with each chaincode. For each chaincode, we initially populate the world state as described below. We intentionally used small numbers of keys in order to induce a high number of conflicts.

[1] Fabric 0.6 follows an Order-Execute model which is incompatible with the Execute-Order-Validate model of Fabric 1.4. As of April 2020 Blockbench supports Fabric 1.4.

Table 2: Chaincode functions and operations

EHR

Functions	Operations	Functions	Operations
initLedger	2xW	addEhr	2xR, 2xW
grantProfileAccess	1xR, 1xW	readProfile	1xR
revokeProfileAccess	1xR, 1xW	viewPartialProfile	1xR
revokeEhrAccess	2xR, 2xW	viewEHR	1xR
grantEhrAccess	2xR, 2xW	queryEHR	1xR

DV

Functions	Operations
initLedger	3xW
vote	1xR, 2xRR, 2xW
closeElctn	1xR, 1xW
qryParties	1xR, 1xRR
seeResults	1xR, 1xRR

SCM

Functions	Operations
initLedger	2xW
pushASN	1xW
Ship	2xR, 2xW
Unload	2xR, 2xW
queryASN	1xRR
queryStock	1xRR*

DRM

Functions	Operations
initLedger	2xW
create	1xR, 2xW
play	2xR, 1xW
queryRghts	2xR
viewMetaData	1xR
calcRevenue	1xRR*

*Fabric does not detect phantom reads for certain type of range reads (RR)

Electronic Health Records (EHR): This chaincode manages medical health records provided by medical institutions or other service providers. Every patient owns two entities, its profile (personal information) and its electronic health records. We generate 100 profiles and 100 electronic health records to populate the world state. Access to either or both can be granted or revoked at any time. If access is granted, medical actors (doctors or researchers) may query or update the records. This chaincode only deals with the access credentials and logical connections. The actual data can be stored off chain. Similar designs are available in the literature [3, 27, 28, 51].

Digital Voting (DV): A predefined set of 1000 voters and 12 competing parties participate in this digital voting scenario. Votes may be cast only during the election phase which ends with a close transaction. A voter is blocked from casting multiple votes. Querying the list of parties and counting out the votes are also included. The `qryParties` and `seeResults` functions query all 12 parties and the `vote` function queries all 1000 voters. This design is based on the work of Yavuz et al. [52].

Supply Chain Management (SCM): This chaincode implements the standard operations of a general logistics network. Logistic service providers (LSP) and logistic units can be managed. For our workload, we generate five LSPs where four LSPs have 400 logistic units each and the fifth LSP has 800 logistic units. Global trade item numbers and serial shipping container codes are used to track the logistic units which can be single trade items or a group of items. An advanced shipping notice (ASN) can also be defined prior to a shipping. Upon successful shipping, the logistic unit is removed from the originating LSP and added to the destination LSP. Information regarding the units located at any LSP or information about a specific logistic unit can be retrieved. Also, any logistic unit can be unloaded to extract the embedded trade items. The `queryASN` function queries all the logistic units of a random LSP. This chaincode is based on the concepts of Perboli et al. [34].

Digital Rights Management (DRM): This chaincode allows artists to share and manage their work on a blockchain. The metadata of 200 artworks is stored in the dot blockchain media format [13] and 200 right holders can be identified with the industry standard IDs [8]. Metadata and royalty management is handled on the blockchain. The current revenue of the right holders can also be calculated. There are similar blockchain applications in the market that handle music management and distribution [48, 50].

4.4 Synthetic Chaincodes & Workloads

In order to run controlled experiments and microbenchmarks, we developed a chaincode and workload generator. The chaincode generator takes as input the total number of chaincode functions and for each function the number of read, insert, update, delete and range read actions. Users can also input the kind of database they wish to deploy; in case it is CouchDB, the user can also select to include rich queries in the chaincode functions. The final output is a syntactically correct chaincode with the user-specified chaincode functions. The workload generator takes as input the number of transactions, the transaction distribution (percentage of read, insert, delete, update and range read) and the key distribution (Zipfian skew). The output is a set of transactions based on the transaction and key distribution. In this paper, since we need to evaluate transaction failures caused by every transaction type, we generated a chaincode named *genChain* which comprises equally distributed read, insert, update, delete and range read functions. We initialized the world state with a large number of keys (100,000 keys) to run experiments with reduced transaction conflicts and generated read-heavy (RH), insert-heavy (IH), update-heavy (UH), delete-heavy (DH) and range-heavy (RaH) workloads. Each "x"-heavy (where x=read, insert, update, delete and range) workload contains 80% of "x" transactions and a uniform distribution of the four other types of transactions. The range queries access a range of 2, 4 or 8 keys uniformly at random. We also generated a uniform workload of read and update transactions with 3 different key distributions (Zipfian skew: 0, 1, 2).

Table 3: Default values of control variables

Variable	Value	Variable	Value
Fabric Version	Fabric 1.4	Database Type	CouchDB
Chaincode	EHR	Block Size	100
Policy	P_0	Tx arrival rate	100 tps
No. of orgs	2 (C1); 8 (C2)	No. of peers / org	2 (C1); 4 (C2)
Zipfian Skew	1	Workload	Uniform

4.5 Control Variables & Metrics

We run our experiments using four different builds of Fabric. Fabric 1.4 [21] was the latest publicly available release version of Fabric during the inception of this paper. Fabric++ [44], Streamchain [24] and FabricSharp [43] are extensions of Fabric that realize different optimization techniques. The combined number of transactions sent per second from all clients is defined as the *transaction arrival rate* of the system. *Block size* is the number of transactions to be included in one block by the ordering service. We run experiments on the different *chaincodes* (EHR, DV, SCM, DRM and genChain) as explained in Section 4.3 and 4.4. The *database type* can be set as CouchDB, which supports rich queries on values modeled as JSON data, or LevelDB, which is the default database for Fabric.

The *number of organizations*, *number of peers per organization* and the *endorsement policy* are other control variables. The different *endorsement policies* we used are shown in Table 5. We also vary the *workload* to be read heavy, read-write heavy or uniform by changing the number of times each chaincode function is invoked. The keys accessed in our experiments follow a Zipfian distribution [36] with varying *Zipfian skew* values. A Zipfian skew of 0 implies that the keys are accessed uniformly. A positive Zipfian skew implies that the keys are accessed more from the higher range of the set of keys. We also do a network emulation with Pumba [38], which is a chaos testing and network emulation tool for Docker containers, and induce a *network delay* to evaluate different network conditions. Table 3 shows the control variables and their default values.

The performance metrics are collected by parsing the blockchain after each experiment. Therefore, the metrics collection process has no impact on the performance during the experiment. All failures are represented as percentages. All failures including endorsement policy failures are detected only in the validation phase of Fabric. The clients do not resend any failed transactions and both failed and successful transactions are committed to the blockchain. The three extensions to Fabric have some exceptions to this default flow, which are mentioned in the corresponding sections. *Average total transaction latency* is the average of the time taken for each complete transaction flow (all three phases of the E-O-V model) of both failed and successful transactions. *Committed transaction throughput* is the number of transactions committed to the blockchain divided by the total time taken.

5 Experimental Results

We conducted a total of 970 experiments. Each experiment was repeated at least 3 times and the average result is presented here. Transactions were sent from all clients for a duration of three minutes for every experiment. With both the cluster configurations, our testbed yields a throughput of 200 tps (transactions per second). The default values of the control variables are defined in Table 3 and changes to any of these values are explicitly mentioned with the results. All the below referenced results, not included in this paper, as well as larger graph renderings, are available in an extended technical report [7].

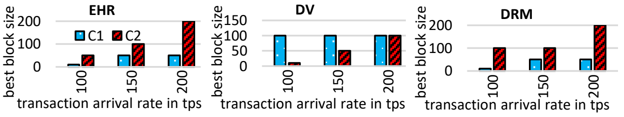


Figure 4: Best block size at different transaction arrival rates

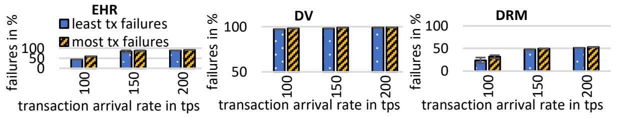


Figure 5: Minimum and maximum transaction failures

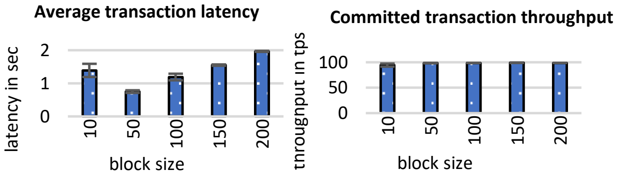


Figure 6: Latency and throughput at different block size

5.1 Results for Fabric 1.4

5.1.1 *Block size and transaction arrival rate*:

(a) Transaction Failures: We define the *best block size* as the block size at which there is the least percentage of failed transactions and the *worst block size* as the block size at which the percentage of failed transactions is maximum. Figure 4 shows the *best block size* at different transaction arrival rates for different chaincodes and cluster setups. Figure 5 shows the maximum and minimum percentage of failed transactions corresponding to the *best block size* and *worst block size* on the C2 cluster. Figure 6 shows the effect of block size on latency and throughput at an arrival rate of 100 tps for the EHR on the C2 cluster.

Observations: For all the chaincodes, we see an approximately linear relation between increasing transaction arrival rate and the best block size. There is up to 60% decrease in failed transactions between the *worst block size* and the *best block size*. For example, the DRM chaincode at 50 tps yielded 21.14% failures with the *worst block size* while we observed only 8.07% failures with the *best block size* (not shown in the figure).

At low transaction arrival rates, a low block size is preferable, so that the ordering service does not have to wait for a long time until enough transactions have arrived for a block to be created. However, as the transaction arrival rate increases, this delay becomes less significant. Hence, at higher transaction arrival rates, larger blocks can be built. Building larger blocks has the advantage that there is less overhead involved in the ordering service and in the validating peers. This reduces the chance of temporary overload and queuing, leading to lower transaction latency and, hence, to less MVCC read conflicts. We can also observe that due to more resources the C2 cluster setup supports higher block sizes at high transaction rates than the C1 cluster setup. To further understand the effect of block size, we also analyzed the latency and throughput. The *best block size* for EHR at 100 tps is 50, the latency for EHR is lowest at a block size of 50 and the throughput is not significantly affected by block size (Figure 6). Similarly, the *best block size* at 150 tps is 100 and the highest throughput and least latency are also for block size 100 (not shown in the figure). So, the block size where other performance metrics such as latency and throughput have better values is also where the failures are least. We made similar observations with the other chaincodes at different transaction arrival rates.

The number of MVCC read conflicts also depends on the number of keys to validate which varies depending on the chaincode functions. Hence, the best setting of the block size with increasing transaction arrival rate is different for different chaincodes. Three of the five chaincode functions in DV have range queries which cause a higher failure rate when compared to the other chaincodes. We can still observe the influence of block size on transaction failures, but the effect is less significant than for the other chaincodes.

Implications: The percentage of failed transactions that occur when the transaction arrival rate changes depends on the block size in most cases. This dependency changes with different chaincodes and cluster setups. Determining this dependency for each chaincode on a specific Fabric network and adapting the block size when the arrival rate changes is an efficient and simple approach to reduce transaction failures. Further, the measured latency and throughput comprise of both successful and failed transactions. So having low latency or high throughput is irrelevant if the transaction failure rate is very high. Therefore, transaction failures should be analyzed along with latency and throughput to ensure good performance.

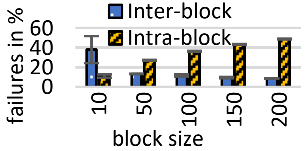


Figure 7: Effect of block size

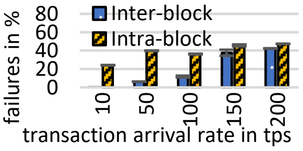


Figure 8: Effect of load

(b) MVCC read conflicts: We evaluate the number of inter-block MVCC read conflicts and intra-block MVCC read conflicts with changing block size and transaction arrival rate in Figures 7 and 8, respectively, for the EHR chaincode on the C2 cluster.

Observations: The number of intra-block MVCC read conflicts increases when the block size increases because when more transactions are included in a block, there is a higher chance of dependencies between transactions which lead to conflicts. Conversely, with increasing block size the inter-block MVCC read conflicts decrease because the conflicts are more likely to have already occurred within the block than across blocks. Also, both failures increase with increasing transaction arrival rate. Further, the *best block size* for EHR at 100 tps is 50, and the sum of both inter and intra-block MVCC read conflicts are low at this block size (Figure 7). But if we consider them individually, inter-block conflicts are least at block size 200 while intra-block conflicts is least at block size 10.

Implications: Changing the block size with respect to any one type of failure is not useful since different types of failures have a different relation to block size. A trade-off between the least inter-block and intra-block MVCC read conflicts is necessary to ensure that the total transaction failures are least.

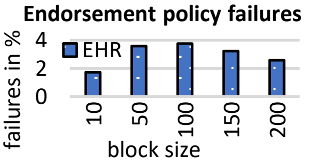


Figure 9: Endorsement policy failures

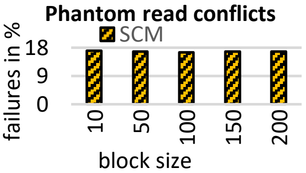


Figure 10: Phantom read conflicts

(c) Endorsement policy failures and phantom reads: Figure 9 and Figure 10 show the effect of block size on endorsement policy failures for the EHR chaincode and phantom reads for the SCM chaincode, respectively, on the C2 cluster.

Observations: Since endorsement policy failures are caused by inconsistent world states, block size does not have a significant impact. A single range query transaction can have a dependency with multiple other transactions within and across blocks that write to at least one key contained in the range.

Implications: Though adapting the block size can help in reducing transaction failures, some types of failures like endorsement failures and phantom reads are not affected.

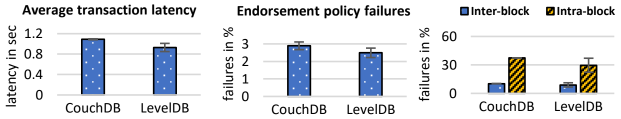


Figure 11: Effect of database type on latency and failures

Table 4: Effect of database type

Avg tx latency (s)			Tx failures (%)			Func call latency (ms)		
Workload	**CDB**	**LDB**	**Workload**	**CDB**	**LDB**	**Function**	**CDB**	**LDB**
RH	18.04	3.22	RH	5.65	1.38	Get	8.3	0.6
IH	18.34	7.93	IH	2.17	1.36	Put	0.8	0.5
UH	20.82	9.86	UH	31.31	23.03	Range	88	1.4
RaH	101.63	4.14	RaH	34.18	5.19	Delete	1.2	0.6
DH	18.48	1.22	DH	1.11	0.18			

5.1.2 Database type: The effect of using CouchDB (CDB) or LevelDB (LDB) with uniform workload and the EHR chaincode is shown in Figure 11 and the results with different workloads with the *genChain* chaincode are shown in Table 4. It also shows the latency of the different chaincode function calls on both databases.

Observations: LevelDB performs better across different chaincodes and workloads. To better understand the overhead of CouchDB, we further analyze the latency of each different function call in the chaincode (Table 4). Latency is lower when using LevelDB because it is embedded with the peer process whereas CouchDB is an external database invoked via REST APIs [9]. The percentages of endorsement policy failures and MVCC read conflicts are also lower with LevelDB. A lower latency implies that transactions are committed faster. This leads to fewer conflicts between transactions, i.e., MVCC read conflicts are reduced. Further, the world state can be updated faster which leads to a slight reduction of endorsement policy failures. Also, the latency and correspondingly the number of failures for a range-heavy workload is significantly higher for CouchDB (Table 4). This is because the entire range of keys are read from the database during the endorsement phase and validation phase to ensure that no key has changed between the phases (phantom read detection). For an external database such as CouchDB, this induces a significant overhead.

Implications: CouchDB supports rich queries such as sort and filter which are useful for many use cases, while LevelDB only supports simple get and set queries. Though CouchDB has richer functionality, our results show that it affects the performance of Fabric. So, if a chaincode can be designed without rich queries, it is always better to use LevelDB to reduce transaction failures. Also, rich queries supported by CouchDB can provide similar functionality as a range query, but Fabric does not re-execute a rich query in the final validation phase and therefore, provides no guarantees on the validity of the query result (no phantom read detection) [45]. So the user needs to make a trade-off between performance and query result validity.

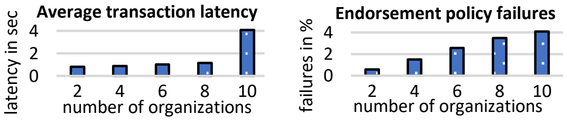


Figure 12: Effect of the number of organizations

5.1.3 Number of organizations: Figure 12 shows the effect of the number of organizations on latency and transaction failures. Experiments were conducted with different numbers of organizations (2, 4, 6, 8 and 10) on the C2 cluster. There are four peers per organization and therefore increasing the number of organizations increases the number of peers involved in the system.

Observations: The transaction latency and the endorsement policy failures increase with the number of organizations. When there are more peers, the number of replicas of the world state increases, which in turn increases the possibility of inconsistent world states between the peers which causes the observed increase in the number of endorsement policy failures.

Implications: The number of organizations and peers that form a Fabric network should be restricted as much as possible to reduce endorsement policy failures. For example, geographically close or functionally similar branches of a company could be considered as a single organization. The trade-off between performance and inclusion of more participants should be considered when building a Fabric network.

Table 5: Endorsement Policies

P_0: "N-of": ["signed-by": 0 , ..., "signed-by": N-1]
P_1: "2-of": ["signed-by": 0, "1-of": ["signed-by": 1 , ..., "signed-by": N-1]]
P_2: "2-of": ["1-of": ["signed-by": 0 , ..., "signed-by": N/2],
"1-of": ["signed-by": N/2+1 , ..., "signed-by": N-1]]
P_3: "(N/2+1)-of": ["signed-by": 0 , ..., "signed-by": N-1]
N: number of organizations, "n-of": n signatures required
An "n-of" clause nested inside another "n-of" is called a sub-policy.

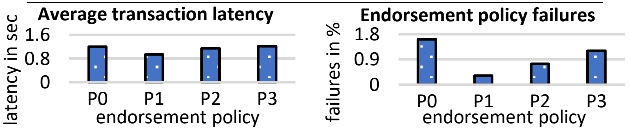


Figure 13: Effect of endorsement policies

5.1.4 Endorsement Policy: Figure 13 shows the impact of different endorsement policies (cf. Table 5).

Observations: The number of endorsement policy failures is maximal when the most endorsement signatures are required. Endorsement policy P_0 requires N signatures and P_3 requires a quorum $(N/2 + 1)$ of signatures. When more endorsements are required, the world state needs to be consistent on a greater number of peers. Therefore, there is a higher chance for endorsement policy failures.

Even when an equal number of signatures is required, an increase in the number of sub-policies induces an increasing number of failures. For instance, P_1 and P_2 both require two signatures. P_1 requires one signature from organization Org_0 and one signature from any of the other organizations, whereas P_2 requires one signature from the first half of the organizations and one signature from the second half of the organizations. Therefore, P_1 includes one sub-policy while P_2 includes two sub-policies. Consequently, the number of endorsement policy failures in P_2 is higher. The endorsement policy is parsed during the VSCC validation and compared with the endorsement signatures of a transaction. Each sub-policy is a separate search space, so that the time taken for validation increases with an increasing number of sub-policies, which in turn also increases the chance of endorsement policy failures. At the same time, the average transaction latency increases.

Implications: Similar to restricting the number of organizations and peers participating in the network, enterprises should also restrict the number of participants in the endorsement policy. For example, if one organization has a higher decision-making power or is more trustworthy than another, then one can reconsider if really both organizations are required to endorse the transactions. Also, one can consider simplifying endorsement policies, such that the number of sub-policies is reduced. For example, the policy:

"4-of": ["2-of": [*Org₀*, *Org₁*], "2-of": [*Org₂*, *Org₃*]]

can also be written as:

"4-of": [*Org₀*, *Org₁*, *Org₂*, *Org₃*]

In either formulation, all four organizations have to endorse the transactions.

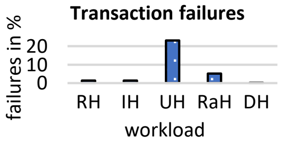


Figure 14: Effect of workload

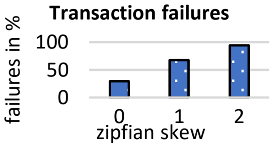


Figure 15: Effect of key skew

5.1.5 Workload: Figure 14 shows the effect of different workloads on failures with *genChain* chaincode on the C2 cluster.

Observations: Insert-heavy and delete-heavy workloads insert and delete unique keys, thus avoiding transaction conflicts. Hence, these workloads have the least failures. Read-heavy and range-heavy workloads induce lower failures compared to update-heavy workloads because only write transactions cause dependencies with reads that lead to conflicts. This result is independent of block size.

Implications: If the use case permits, one should aim to batch read-only transactions together to ensure they all succeed. Our results also back the recommendation by Fabric [22] to not submit read-only transactions for ordering and validation since the necessary result is already delivered after the execution phase itself. It is only necessary to submit read-only transactions if one needs a record on the blockchain for auditing purposes.

5.1.6 Zipfian skew: Figure 15 shows the effect of the Zipfian skew for key access with *genChain* chaincode and a uniform workload.

Observations: When the key access is more skewed, the percentage of failures increases. The number of conflicts will increase if more transactions access the same (set of) key(s).

Implications: Chaincodes and the database structure can be designed such that key access is less skewed. For example, in the EHR chaincode, the `addEHR` function uses a `PatientID` as the key to add any new medical record for a patient. One could replace `patientID` by two new keys `PatientID_XrayID` and `PatientID_MRIID` such that a transaction updating a patient's Xray and another transaction updating the same patient's MRI will not conflict with each other. Overall, modelling the data representation is an important aspect that should be carried out carefully.

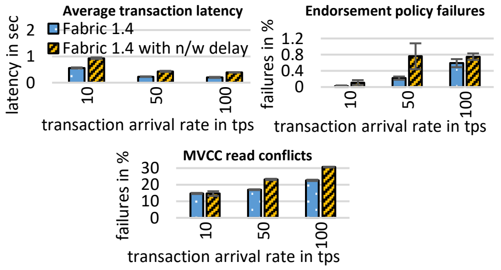


Figure 16: Fabric 1.4 with and without network delay

5.1.7 Network delay: Figure 16 shows the effect of an induced additional network delay of 100±10 ms for one organization to emulate the scenario of a geographically distributed organization.

Observations: The additional network delay causes an increase in the transaction latency and consequently increases the number of failures. The endorsement policy failures are affected because the network delay increases the chance of inconsistent world states between peers. MVCC read conflicts are affected because the time between endorsement and validation of a transaction is increased.

Implications: Endorsement policy failures are highly affected by network delays. This implies that if several geographically far apart organizations are part of the endorsement policy, the number of endorsement policy failures will increase. Hence, network delays must be considered in the design of the Fabric network and of the endorsement policies. Finally, MVCC read conflicts that are caused by large network delays are inherent to the optimistic concurrency control in Fabric and cannot be avoided.

5.2 Results for Fabric++

Sharma et al. [44] designed an optimized extension of Fabric which can effectively resolve intra-block MVCC read conflicts. In the ordering phase of Fabric++, a conflict graph is generated for the transactions in a block and all cycles in the graph are identified. Cycles are removed from the graph by aborting transactions in the ordering phase itself. The resulting acyclic graph of transactions is serialized and sent to the validation phase.

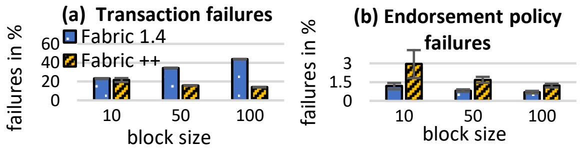


Figure 17: Effect of block size

5.2.1 Block Size: Figure 17 (a) compares the percentage of transaction failures for Fabric 1.4 and Fabric++ at different block sizes.

Observations: At a fixed transaction arrival rate with Fabric++, the transaction failures decrease with increase in block size whereas the trend is reverse for Fabric 1.4. A larger block size gives more reordering possibilities for Fabric++ which leads to fewer failures.

Implications: To efficiently utilize the transaction reordering implemented by Fabric++, use a larger block size.

5.2.2 Endorsement policy failures: Figure 17 (b) compares the percentage of endorsement policy failures in Fabric 1.4 and Fabric++.

Observations: The endorsement policy failures are higher for Fabric++ because there are less transaction failures due to MVCC read conflicts. Hence, the rate of updates of the world state is higher, because only successful transactions are committed to the world state. As the rate of updates is higher, there is also a higher chance of inconsistencies between the world state replicas on the different peers, which leads to more endorsement policy failures.

Implications: Reordering cannot resolve endorsement policy failures. This problem has to be investigated and treated separately, e.g., by reconsidering the endorsement policy or the design of the Fabric network.

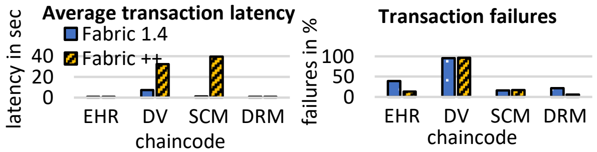


Figure 18: Effect of different chaincodes on Fabric++

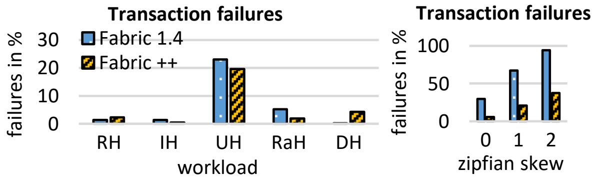


Figure 19: Effect of different workloads on Fabric++

5.2.3 Chaincodes and Workloads: Figures 18 and 19 compare the latency and percentage of transaction failures in Fabric 1.4 and Fabric++ with different chaincodes, workloads and key distribution.

Observations: The total failures do not significantly decrease with Fabric++ when evaluated with the DV and SCM chaincodes. These two chaincodes include range queries that cause phantom read conflicts. Each range query involves a large range of keys (800 to 1000 keys), so that transactions will have dependencies on multiple other transactions. The latency is significantly higher for Fabric++ with these chaincodes. Fabric++ creates conflict graphs and then makes them acyclic to resolve transaction conflicts. To this end, they approximate a solution to the Minimum Feedback Vertex Set (MFVS) problem (which is an NP-hard problem). Because of the high number of dependencies induced by range queries, generating the conflict graph and reordering the range queries become very time consuming in Fabric++. With the *genChain* chaincode, Fabric++ reduces transaction failures for most of the workloads. Since the range queries in the range-heavy workload have a smaller range (2, 4 and 8), we observe a reduction in failures even in the presence of range queries. Further, Fabric++ does not have a positive effect on read-heavy and delete-heavy workloads because although the reordering possibilities are few for these workloads, the reordering process is still being executed and this increases the latency.

Implications: Fabric++ [44] did not evaluate the system in the presence of large range queries. Therefore, our results provide new insights on the impact of range queries on Fabric++. If the use case permits, one should consider designing chaincodes with smaller range queries when using Fabric++. Fabric++ could be optimized in the future to handle range queries more efficiently, e.g., by using a different algorithm to tackle the reordering problem. Also, the reordering potential of a workload needs to be analyzed to efficiently use Fabric++.

5.3 Results for Streamchain

Streamchain [24] is an extension of Fabric that focuses on reducing the latency by sending transactions one-by-one instead of creating a block. In the validation phase, parallel validation of signatures and pipelining are implemented. The current prototype requires that the ledger and the world state are stored on a RAM disk both in the ordering service and the peers.

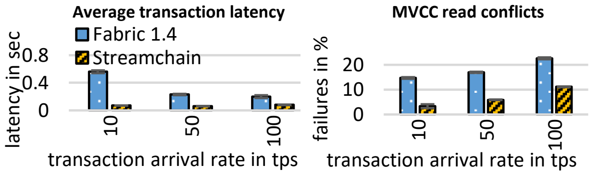


Figure 20: Comparison of Streamchain and Fabric 1.4

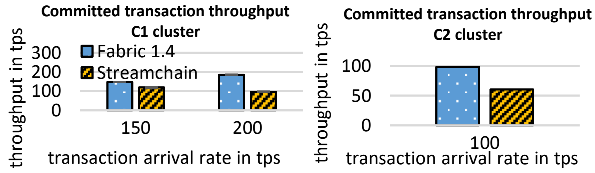


Figure 21: Latency and committed transaction throughput

5.3.1 Latency and transaction failures: Figure 20 compares the performance of Streamchain and Fabric 1.4 at transaction rates of 10, 50 and 100 tps. Figure 21 shows the committed transaction throughput for both at higher arrival rates of 150 and 200 tps on the C1 cluster and 100 tps on the C2 cluster. Fabric 1.4 is set with a block size of 10 (we observed similar results with block sizes 50 and 100).

Observations: The latency and transaction failures are lower for Streamchain up to a transaction arrival rate of 100 tps on the

C1 cluster and up to 50 tps on the C2 cluster. Since the transactions are streamed one-by-one and stored on a RAM disk, the world state is updated quickly, thus reducing the MVCC read conflicts. Since the latency is lower, endorsement policy failures also reduce slightly (not shown in the figure). Beyond a transaction arrival rate of 150 tps on the C1 cluster, Streamchain does not provide enough throughput to handle the load. Further, on the larger C2 cluster the overhead is prominent even at an arrival rate of 100 tps. Streaming the transactions one-by-one will increase the communication overhead between the orderer and the multiple peers. At higher transaction rates and with larger number of peers (C2 cluster), this results in queuing of transactions.

Implications: Streaming the transactions one-by-one helps to update the world state faster at low transaction arrival rates. But Streamchain needs to be further optimized to handle high transaction arrival rates and scaling.

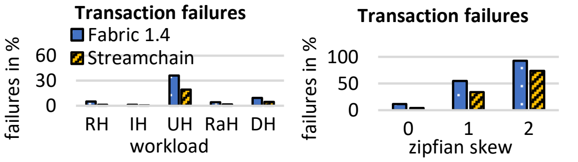


Figure 22: Effect of different workloads on Streamchain

5.3.2 Workloads: Figure 22 compares the performance of Streamchain and Fabric 1.4 with different workloads and key distribution at 50 tps on the C2 cluster.

Observations: Streamchain reduces the transaction failures regardless of the type of workload or key distribution. This is because the optimization used by Streamchain (streaming transactions one-by-one) is independent of the type of transaction.

Implications: Failures are always reduced regardless of the type of the workload or key distribution.

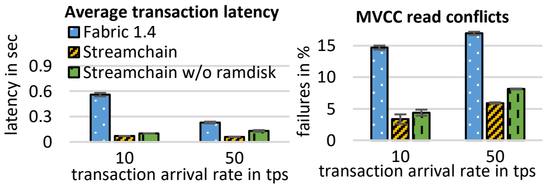


Figure 23: Streamchain with and without a RAM disk

5.3.3 Effect of RAM disk storage: Figure 23 compares the performance of Streamchain with and without a RAM disk.

Observations: Streamchain with RAM disk performs better than without RAM disk. This is an expected result, as the RAM disk allows for faster reads and writes. At lower transaction rates, the latency and MVCC read conflicts of Streamchain are improved compared to Fabric, even if there is no RAM disk used. However, at a transaction rate greater than 50 tps, the throughput of Streamchain without RAM disk was too low to sustain the workload, bringing the system into an unstable condition (not shown in the figure). Streamchain cannot handle the streaming of transactions one-by-one without a fast storage at higher transaction rates.

Implications: The performance improvements of Streamchain are to a large extent caused by the use of a RAM disk storage. The authors of Streamchain have proposed the concept of a virtual block boundary which could be used to commit transactions as blocks while still streaming transactions one-by-one in the ordering service. This concept, if implemented, could potentially remove the need for a RAM disk storage.

5.4 Results for FabricSharp

Ruan et al. [43] designed an optimized extension of Fabric which can effectively resolve MVCC read conflicts. Similar to Fabric++ [44], FabricSharp also generates conflict graphs and serializes them. Transactions which cannot be serialized will be aborted before the ordering phase. But unlike Fabric++, FabricSharp generates the conflict graph across blocks and therefore handles both inter-block and intra-block MVCC read conflicts.

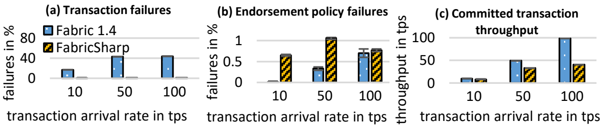


Figure 24: Comparison of FabricSharp and Fabric 1.4

5.4.1 Transaction failures: Figure 24 (a) & (b) compares the performance of FabricSharp and Fabric 1.4 at different arrival rates.

Observations: The transaction failures are significantly lower for FabricSharp. Since all transactions are serialized there are no MVCC read conflicts. FabricSharp does not support range read queries and therefore there are no phantom reads. So only endorsement policy failures are observed for FabricSharp. In FabricSharp, the execution and validation phase are parallelized by using block snapshots at the start of the execution phase. This can introduce stale snapshots that result in more endorsement policy failures.

Implications: FabricSharp is highly effective in resolving MVCC read conflicts but does not resolve endorsement policy failures.

5.4.2 Throughput: Figure 24 (c) compares the throughput of FabricSharp and Fabric 1.4 at 10, 50 and 100 tps.

Observations: The committed transaction throughput is lower for FabricSharp. This is an expected result since FabricSharp aborts non-serializable transactions before the ordering phase and only commits successful transactions (and endorsement failures).

Implications: On the one hand, FabricSharp updates the blockchain with only the successful transactions, thus reducing the overhead in the validation phase, but on the other hand, there is no record of failed transactions on the blockchain which could be useful for debugging and auditing purposes.

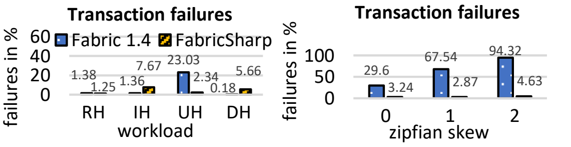


Figure 25: Effect of workload and skew on FabricSharp

5.4.3 Workloads: Figure 25 compares FabricSharp and Fabric 1.4 with different workloads and skew on the C2 cluster with the *genChain* chaincode. We do not use the range-heavy workload because range queries are not supported by FabricSharp.

Observations: FabricSharp significantly reduces failures with update-heavy workloads. But FabricSharp does not have a positive effect on insert-heavy and delete-heavy workloads since insert and delete transactions access unique keys which have no dependencies with other transactions. Thus, reordering in FabricSharp can only resolve a limited number of conflicts for these workloads, while the overhead of reordering actually increases the number of failures.

Implications: The reordering potential of a workload needs to be analyzed before adopting FabricSharp. This observation is similar for Fabric++ (Section 5.2.3).

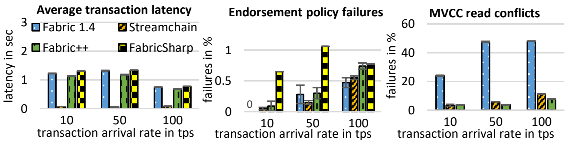


Figure 26: Comparison of Fabric systems on C1 cluster

5.5 Comparison of Fabric-like systems

Figure 26 compares the latency and transaction failures of all the Fabric-like systems with the EHR chaincode.

Observations: We can observe that Fabric++ and FabricSharp have a similar transaction latency as Fabric 1.4, while Streamchain has a significantly lower latency. We also observe that all three optimizations of Fabric show a significant reduction in the number of failures, but none of them resolve endorsement policy failures.

Implications: FabricSharp has the best optimization technique to reduce transaction failures when compared to Fabric++ and Streamchain, although it reduces the committed transaction throughput (cf. Section 5.4.2). The effect of range queries on FabricSharp and Streamchain remains to be studied. Streamchain reduces the latency far better than Fabric++ and FabricSharp, although this is partly due to the use of a RAM disk storage as explained in Section 5.3.3. While all three optimizations, Fabric++, Streamchain and FabricSharp, work well in this setting (low transaction arrival rates, no range queries), our previous experiments have revealed some of their limitations.

6 Lessons Learned

In this section, we summarize the insights we gained from our experiments, explain with examples how these can be leveraged by a Fabric user and discuss future research directions.

6.1 Insights & Recommendations

Types of failures: The three types of failures described in Section 3 are influenced differently by different parameters. Block size has inverse effects on inter-block and intra-block MVCC read conflicts, while it has almost no effect on phantom reads and endorsement policy failures. The number of organizations and endorsement policies have a significant impact on endorsement policy failures while they have insignificant influence on other failures. No parameter tuning in Fabric 1.4 could reduce phantom reads, but reordering the transactions using Fabric++ reduced them. However, Fabric++ could not handle large range reads. FabricSharp early aborts all MVCC read conflicts, but does not resolve endorsement policy failures. Both Fabric++ and FabricSharp increase the number of failures with workloads where reordering possibilities are few, while Streamchain reduces failures regardless of the type of workload. However, Streamchain has a high overhead and lower throughput. Based on our observations, we advice users to analyze their use case, workload and also possibly simulate their network and detect the frequency of different types of failures, before tuning the parameters and using the different Fabric optimizations.
Example: If the probability of MVCC read conflicts is high, use Fabric++ or FabricSharp. But if there are very few conflicts these systems will have a negative impact. If phantom reads are high and the range queries have a small range, Fabric++ should be adopted, while for large range queries this will not have an effect. Streamchain should be chosen only if the network expects very low traffic.
Block size: The block size has a significant effect on the number of transaction failures. The best setting depends on the transaction arrival rate and the chaincode. It is a good practice to adapt the block size to the arrival rate.
Example: For the SCM chaincode, we could assume that the holiday seasons would experience high transaction arrival rates if the sales increase. Hence, during those times, change the block size to a higher value. Similarly, during off season, decrease the block size.
Number of organizations & endorsement policies: Transaction failures increase when the number of organizations and the required number of endorsement signatures increase. Also, sub-policies lead to more failures. So, it is a good practice to lower the number of organizations and create simpler endorsement policies.
Example: For the SCM chaincode, there will be multiple LSPs, but all of them need not be separate organizations of the Fabric network. LSPs starting from the same source or travelling to the same destination could be grouped together. Also, the LSP organizations need not be a part of the endorsement policy since they are only providing a service. Only the owners and stakeholders need to be part of the endorsement policy.
Chaincode design & database type: LevelDB shows better performance than CouchDB. Users should try to design chaincodes that do not require rich queries, so that LevelDB can be used. Also, since phantom reads are not effectively resolved with any of the Fabric systems, range queries should also be avoided when possible.
Example: In the DRM chaincode, a range query is used to query the play count of all the music owned by a specific artist and calculate the total revenue. Instead, every time a song is played, a unique key for each artist to calculate the total revenue could be incremented. This way, a range query could be completely avoided.
Client design: Read-heavy workloads show lower failures. So, depending on the use case, read-only transactions can be batched at the client side and submitted together. Also, clients can be designed to identify read-only transactions and avoid submitting them to the ordering service since the result of the query is already obtained. Fabric also provides event services [33] that can be used to update an off-chain database which could be used for read purposes.
Example: In the SCM chaincode, one needs to read the blockchain multiple times for auditing purposes. It might also be essential to submit these read transactions to the blockchain to keep track of the auditing process. In this scenario, it would be ideal to batch these reads together and submit them when it is not a peak time for other logistic-relating transactions.

Our 4 main recommendations for Fabric users are:

(1) Monitor the trend of transaction arrival rates and adapt the block size at appropriate times.
(2) Design a Fabric network with fewer organizations, fewer endorsement signatures and fewer endorsement sub-polices.
(3) While designing the chaincode, avoid rich queries and range queries unless they are absolutely necessary.
(4) Avoid the submission of read-only transactions to the ordering service or batch them together for submission.

6.2 Future research directions

Adaptive block size: A constant block size is not ideal when the transaction arrival rate changes. The ideal block size for various chaincodes is also different. This establishes the need for a dynamically changing block size. Since the transaction arrival rate cannot be determined beforehand and the dependency between arrival rate and block size changes for different chaincodes, it would be useful to monitor the system and adapt the block size dynamically. There are already adaptive blockchain systems that focus on storage or security [10, 40], but not transaction failures.

Database optimization: There is a clear decrease in performance with CouchDB; however, many chaincodes require the use of rich queries. A productive research focus would be to optimize CouchDB or integrate other databases to reduce commit latency in the peers.

Reduce endorsement policy failures: Inconsistency of world states is a well-known problem and there is already research in this direction [26, 53]. It would be an interesting approach to integrate such research with the Fabric framework and observe the effects on endorsement policy failures.

Chaincode optimizations: There is very little research on designing Fabric chaincodes. A challenging research direction would be to analyze different Fabric chaincodes and derive optimization techniques that can reduce transaction failures.

7 Related Work

Dinh et al. [12] and Pongnumkul et al. [35] present a comparative study of different blockchain frameworks including Fabric but both are based on Fabric version 0.6 which followed an Order-Execute (O-E) design model based on PBFT consensus. The current version of Fabric follows the E-O-V model (cf. Section 2) and only supports a crash-fault tolerant consensus model. The O-E model and the E-O-V model have significant differences and therefore, the results of these papers are not valid for the current version of Fabric.

Many related papers evaluate the performance of Fabric [1, 4, 47]. We go far beyond these existing evaluations and directly compare a large number of systems (Fabric 1.4, Fabric++, Streamchain, FabricSharp) using a large range of different workloads. Further, our focus is on transaction failures, while existing evaluations are mostly concerned with throughput and latency. While throughput and latency are important performance metrics, they are irrelevant if most transactions fail. Similar to our findings, Thakkar et al. [47] also point out the overhead of CouchDB in terms of latency and throughput. In our work, we further explain this overhead by analyzing the latency of each function call in the chaincode. This way, we found that range queries are particularly expensive with CouchDB; a result that has not been reported in [47].

Goel et al. [15] propose a prioritization-based transaction validation model, but they evaluate neither the type nor the number of conflicts. István et al. [24] introduce the concept of a virtual block boundary that can reduce the staleness of data used to execute new transactions. However, they do not analyze transaction failures. The main goal of Sharma et al. [44] with Fabric++ is to reduce the number of transaction conflicts by using optimization strategies of database-like transaction reordering and early aborts. They evaluate two types of MVCC conflicts, but do not discuss endorsement failures and phantom reads. Further, we identified that the effect of blocksize on transaction failures at a fixed transaction rate is inverse for Fabric 1.4 and Fabric++, which is a new insight. Sharma et al. [44] only employ a fixed transaction rate and two chaincodes; We employ multiple chaincodes and various transaction rates, so that our analysis is much more comprehensive. Ruan et al. [43] designed an extension of Fabric which also generates conflict graphs and serializes them. However, their evaluation does not show the effect of blocksize on failures, but only on throughput and latency.

Gorenflo et al. [16] aim to reduce transaction failures by re-executing the conflicting transactions, but they currently have no implementation or evaluation. Gorenflo et al. [17] improve the throughput of Fabric by using multiple optimization strategies. The evaluation is done with a workload of write-only transactions which will never have MVCC read conflicts. Nasirifard et al. [31] use the concept of conflict-free replicated datatypes (CRDT) to resolve conflicts. However, their approach is only applicable for use cases that can be modelled with CRDTs.

Some of our observations are comparable to research in the database domain. While evaluating SharedDB [14], a query processing system that batches queries and shares computations, the authors observe an increase in latency with increasing batch size. Similarly, in OLTPShare [42], a batching scheme for OLTP workloads, smaller batch sizes reduce the potential of sharing while larger batch sizes introduce high latency. Stonebraker et al. [46] observe that developing an application-specific DBMS improves the performance compared to reusing existing DBMS solutions. These three observations are comparable to some of our findings such as that block size in Fabric has a significant influence on failures and that LevelDB, which is embedded in Fabric, performs better than an external database. Though we can draw such parallels with research in the database field, Fabric follows an optimistic concurrency control model that is significantly different from these DBMSs. Additionally, Fabric has other control parameters such as organizations and endorsement policies which are related to blockchains. Thus, the results and inferences in our paper are novel. Also, our work goes beyond existing database research by analyzing the effect of an extensive set of control variables on transaction failures in different extensions of Fabric and focuses on the distributed processing of transactions.

8 Conclusions

In this paper, we formally defined the different transaction failures that occur in Fabric. We designed our own benchmarking system HyperLedgerLab and conducted extensive experiments to analyze the effects of different parameters on failures. We observed a clear dependency between block size and failures, and the optimal block size induced up to 60% reduction in failures. We also deployed three optimizations of Fabric, Streamchain [24], Fabric++ [44] and FabricSharp [43], on HyperLedgerLab and analyzed their performance. We then derived a set of practical recommendations for Fabric users based on our results and discussed possible future research directions. In the future, we will integrate more chaincodes into HyperLedgerLab and also deploy other Fabric optimizations to exhaustively study and compare them.

Acknowledgments

This work is funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 392214008.

References

[1] 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. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. ACM, New York, NY, USA, Article 30, 15 pages. https://doi.org/10.1145/3190508.3190538

[2] Apache CouchDB 2020. https://couchdb.apache.org/. (2020). [Online; accessed 24-February-2021].

[3] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. 2016. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE, 25–30.

[4] A. Baliga, N. Solanki, S. Verekar, A. Pednekar, P. Kamat, and S. Chatterjee. 2018. Performance Characterization of Hyperledger Fabric. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 65–74. https://doi.org/10.1109/CVCBT.2018.00013

[5] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. 2016. Corda: an introduction. *R3 CEV, August* 1 (2016), 15.

[6] Build the future of Open Infrastructure 2020. https://www.openstack.org/. (2020). [Online; accessed 24-February-2021].

[7] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric (Extended version). (2021). arXiv:2103.04681

[8] CISAC 2021. https://www.cisac.org/services/information-services/ipi. (2021). [Online; accessed 24-February-2021].

[9] CouchDB as the State Database 2020. https://hyperledger-fabric.readthedocs.io/en/release-2.2/couchdb_as_state_database.html. (2020). [Online; accessed 01-October-2020].

[10] Syed Muhammad Danish, Kaiwen Zhang, and Hans-Arno Jacobsen. 2020. BlockAM: An Adaptive Middleware for Intelligent Data Storage Selection for Internet of Things. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. 61–71. https://doi.org/10.1109/DAPPS49028.2020.00007

[11] Dean, J. and Ghemawat. 2020. https://github.com/google/leveldb. (2020). [Online; accessed 24-February-2021].

[12] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 1085–1100. https://doi.org/10.1145/3035918.3064033

[13] Dot blockchain media 2020. https://dotblockchainmusic.com/. (2020). [Online; accessed 24-February-2021].

[14] Georgios Giannikis, Gustavo Alonso, and Donald Kossmann. 2012. SharedDB: Killing One Thousand Queries with One Stone. *Proc. VLDB Endow.* 5, 6 (Feb. 2012). https://doi.org/10.14778/2168651.2168654

[15] Seep Goel, Abhishek Singh, Rachit Garg, Mudit Verma, and Praveen Jayachandran. 2018. Resource Fairness and Prioritization of Transactions in Permissioned Blockchain Systems (Industry Track). In *Proceedings of the 19th International Middleware Conference Industry (Middleware '18)*. ACM, New York, NY, USA, 46–53. https://doi.org/10.1145/3284028.3284035

[16] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. 2019. XOX Fabric: A hybrid approach to transaction execution. *arXiv preprint arXiv:1906.11229* (2019).

[17] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 455–463. https://doi.org/10.1109/BLOC.2019.8751452

[18] Gideon Greenspan. 2015. Multichain private blockchain-white paper. *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf* (2015).

[19] Theo Härder. 1984. Observations on optimistic concurrency control schemes. *Information Systems* 9, 2 (1984), 111 – 120. https://doi.org/10.1016/0306-4379(84)90020-6

[20] Hyperledger Caliper 2020. https://hyperledger.github.io/caliper/. (2020). [Online; accessed 24-February-2021].

[21] Hyperledger Fabric 2020. https://github.com/hyperledger/fabric#releases. (2020). [Online; accessed 24-February-2021].

[22] Hyperledger Fabric Glossary 2020. https://hyperledger-fabric.readthedocs.io/en/release-2.0/glossary.html. (2020). [Online; accessed 24-February-2021].

[23] HyperLedgerLab 2021. https://github.com/MSRG/HyperLedgerLab. (2021). [Online; accessed 24-February-2021].

[24] Zsolt István, Alessandro Sorniotti, and Marko Vukolić. 2018. Streamchain: Do blockchains need blocks?. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. 1–6.

[25] Leslie Lamport. 2001. Paxos made simple. *ACM Sigact News* 32, 4 (2001), 18–25.

[26] Yusen Li and Wentong Cai. 2011. Determining optimal update period for minimizing inconsistency in multi-server distributed virtual environments. In *2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*. IEEE, 126–133.

[27] Tomas Mikula and Rune Hylsberg Jacobsen. 2018. Identity and access management with blockchain in electronic healthcare records. In *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, 699–706.

[28] Daniel-Jesus Munoz, Denisa-Andreea Constantinescu, Rafael Asenjo, and Lidia Fuentes. 2019. ClinicAppChain: A Low-Cost Blockchain Hyperledger Solution for Healthcare. In *International Congress on Blockchain and Applications*. Springer.

[29] Shojiro Muro, Tiko Kameda, and Toshimi Minoura. 1984. Multi-version concurrency control scheme for a database system. *J. Comput. System Sci.* 29, 2 (1984), 207 – 224. https://doi.org/10.1016/0022-0000(84)90031-X

[30] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list at https://metzdowd.com* (03 2009).

[31] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. 2019. FabricCRDT: A Conflict-Free Replicated Datatypes Approach to Permissioned Blockchains. In *Proceedings of the 20th International Middleware Conference (Middleware '19)*. Association for Computing Machinery, New York, NY, USA, 110–122. https://doi.org/10.1145/3361525.3361540

[32] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX Association, Berkeley, CA, USA, 305–320. http://dl.acm.org/citation.cfm?id=2643634.2643666

[33] Peer channel-based event services 2020. https://hyperledger-fabric.readthedocs.io/en/release-1.4/peer_event_services.html. (2020). [Online; accessed 24-February-2021].

[34] Guido Perboli, Stefano Musso, and Mariangela Rosano. 2018. Blockchain in logistics and supply chain: A lean approach for designing real-world use cases. *IEEE Access* 6 (2018), 62018–62028.

[35] Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thajchayapong. 2017. Performance analysis of private blockchain platforms in varying workloads. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–6.

[36] David M. W. Powers. 1998. Applications and Explanations of Zipf's Law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning (NeMLaP3/CoNLL '98)*. Association for Computational Linguistics, USA.

[37] Production-Grade Container Orchestration 2020. https://kubernetes.io/. (2020). [Online; accessed 24-February-2021].

[38] Pumba: Chaos testing tool for Docker 2020. https://github.com/alexei-led/pumba. (2020). [Online; accessed 24-February-2021].

[39] Quorum whitepaper 2016. https://www.blocksg.com/single-post/2017/12/27/Quorum-Whitepaper. (2016). [Online; accessed 24-February-2021].

[40] Shishir Rai, Kendric Hood, Mikhail Nesterenko, and Gokarna Sharma. 2019. Blockguard: Adaptive Blockchain Security. *CoRR* abs/1907.13232 (2019).

[41] Michel Rauchs, Apolline Blandin, Keith Bear, and Stephen B McKeon. 2019. 2nd Global Enterprise Blockchain Benchmarking Study. *Available at SSRN 3461765* (2019).

[42] Robin Rehrmann, Carsten Binnig, Alexander Böhm, Kihong Kim, Wolfgang Lehner, and Amr Rizk. 2018. OLTPshare: The Case for Sharing in OLTP Workloads. *Proc. VLDB Endow.* 11, 12 (Aug. 2018). https://doi.org/10.14778/3229863.3229866

[43] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A Transactional Perspective on Execute-Order-Validate Blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 543–557. https://doi.org/10.1145/3318464.3389693

[44] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the Lines Between Blockchains and Database Systems: The Case of Hyperledger Fabric. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. ACM, New York, NY, USA, 105–122. https://doi.org/10.1145/3299869.3319883

[45] shim - GoDoc 2020. https://godoc.org/github.com/hyperledger/fabric-chaincode-go/shim#ChaincodeStub.GetQueryResult. (2020). [Online; accessed 24-February-2021].

[46] Michael Stonebraker and Ugur Çetintemel. 2005. "One Size Fits All": An Idea Whose Time Has Come and Gone. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*. IEEE Computer Society, USA, 2–11. https://doi.org/10.1109/ICDE.2005.1

[47] P. Thakkar, S. Nathan, and B. Viswanathan. 2018. 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)*. 264–276. https://doi.org/10.1109/MASCOTS.2018.00034

[48] Ujo liberating music: Connecting artists and fans directly using Ethereum 2020. https://ujomusic.com/. (2020). [Online; accessed 24-February-2021].

[49] University of Cambridge Judge Business School: Cambridge Centre for Alternative Finance. 2020. https://www.jbs.cam.ac.uk/faculty-research/centres/alternative-finance/. (2020). [Online; accessed 24-February-2021].

[50] Verifi media: harmonizing media + ownership 2020. https://verifi.media/. (2020). [Online; accessed 24-February-2021].

[51] Guang Yang and Chunlei Li. 2018. A design of blockchain-based architecture for the security of electronic health record (EHR) systems. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE.

[52] Emre Yavuz, Ali Kaan Koç, Umut Can Çabuk, and Gökhan Dalkılıç. 2018. Towards secure e-voting using ethereum blockchain. In *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, 1–7.

[53] Yang Yu, Zhu Li, Larry Shi, Yi-Chiun Chen, and Hua Xu. 2007. Network-aware state update for large scale mobile games. In *2007 16th International Conference on Computer Communications and Networks*. IEEE, 563–568.

[54] Kaiwen Zhang and Hans-Arno Jacobsen. 2018. Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 1337–1346. https://doi.org/10.1109/ICDCS.2018.00134

ACM Publishing License and Audio/Video Release

Title of the Work: Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric
Submission ID:rdm136

Author/Presenter(s): Jeeta Ann Chacko,Ruben Mayer,Hans-Arno Jacobsen

Type of material:full paper

Publication and/or Conference Name: SIGMOD '21: 2021 International Conference on Management of Data Proceedings

1. Glossary

2. Grant of Rights

(a) Owner hereby grants to ACM an exclusive, worldwide, royalty-free, perpetual, irrevocable, transferable and sublicenseable license to publish, reproduce and distribute all or any part of the Work in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library, and to authorize third parties to do the same.

(b) In connection with software and "Artistic Images and "Auxiliary Materials, Owner grants ACM non-exclusive permission to publish, reproduce and distribute in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library.

(c) In connection with any "Minor Revision", that is, a derivative work containing less than twenty-five percent (25%) of new substantive material, Owner hereby grants to ACM all rights in the Minor Revision that Owner grants to ACM with respect to the Work, and all terms of this Agreement shall apply to the Minor Revision.
(d) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

☑ A. Grant of Rights. I grant the rights and agree to the terms described above.

□ B. Declaration for Government Work. I am an employee of the national government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are you a contractor of your National Government? ○ Yes ● No

---

3. Reserved Rights and Permitted Uses.

(a) All rights and permissions the author has not granted to ACM in Paragraph 2 are reserved to the Owner, including without limitation the ownership of the copyright of the Work and all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM in Paragraph 2(a), Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please put the following LaTeX commands in the preamble of your document -

i.e., before \begin{document}:

```
\copyrightyear{2021}
\acmYear{2021}
\setcopyright{acmlicensed}\acmConference[SIGMOD '21]{Proceedings of the
2021 International Conference on Management of Data}{June 18--27,
2021}{Virtual Event , China}
\acmBooktitle{Proceedings of the 2021 International Conference on
Management of Data (SIGMOD '21), June 18--27, 2021, Virtual Event , China}
\acmPrice{15.00}
\acmDOI{10.1145/3448016.3452823}
\acmISBN{978-1-4503-8343-1/21/06}
```

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

If you are using the ACM Interim Microsoft Word template, or still using or older versions of the ACM SIGCHI template, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SIGMOD '21, June 18–27, 2021, Virtual Event , China
© 2021 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8343-1/21/06...$15.00
https://doi.org/10.1145/3448016.3452823

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF to your event contact for processing.

---

4. ACM Citation and Digital Object Identifier.

(a) In connection with any use by the Owner of the Definitive Version, Owner shall

include the ACM citation and ACM Digital Object Identifier (DOI).
(b) In connection with any use by the Owner of the Submitted Version (if accepted) or the Accepted Version or a Minor Revision, Owner shall use best efforts to display the ACM citation, along with a statement substantially similar to the following:

> "© [Owner] [Year]. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in {Source Publication}, https://doi.org/10.1145/{number}."

5. Audio/Video Recording

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ● Yes ○ No

6. Auxiliary Material

Do you have any Auxiliary Materials? ○ Yes ● No

7. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

● We/I have not used third-party material.
○ We/I have used third-party materials and have necessary permissions.

8. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part IV and be sure to include a notice of copyright with each such image in the paper.
● We/I do not have any artistic images.
○ We/I have any artistic images.

9. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants.

10. Enforcement.

At ACM's expense, ACM shall have the right (but not the obligation) to defend and enforce the rights granted to ACM hereunder, including in connection with any instances of plagiarism brought to the attention of ACM. Owner shall notify ACM in writing as promptly as practicable upon becoming aware that any third party is infringing upon the rights granted to ACM, and shall reasonably cooperate with ACM in its defense or enforcement.

11. Governing Law

This Agreement shall be governed by, and construed in accordance with, the laws of the state of New York applicable to contracts entered into and to be fully performed therein.

Funding Agents

1. Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) award number(s):392214008

DATE: **03/01/2021** sent to chacko@in.tum.de at **05:03:48**

Appendix B

How To Optimize My Blockchain? A Multi-Level Recommendation Approach

How To Optimize My Blockchain? A Multi-Level Recommendation Approach

JEETA ANN CHACKO, Technical University of Munich, Germany
RUBEN MAYER*, University of Bayreuth, Germany
HANS-ARNO JACOBSEN, University of Toronto, Canada

Aside from the conception of new blockchain architectures, existing blockchain optimizations in the literature primarily focus on system or data-oriented optimizations within prevailing blockchains. However, since blockchains handle multiple aspects ranging from organizational governance to smart contract design, a holistic approach that encompasses all the different layers of a given blockchain system is required to ensure that all optimization opportunities are taken into consideration. In this vein, we define a multi-level optimization recommendation approach that identifies optimization opportunities within a blockchain at the system, data, and user level. Multiple metrics and attributes are derived from a blockchain log and nine optimization recommendations are formalized. We implement an automated optimization recommendation tool, BlockOptR, based on these concepts. The system is extensively evaluated with a wide range of workloads covering multiple real-world scenarios. After implementing the recommended optimizations, we observe an average of 20% improvement in the success rate of transactions and an average of 40% improvement in latency.

CCS Concepts: • **Information systems → Data management systems**.

Additional Key Words and Phrases: Blockchains, process mining, performance optimization

ACM Reference Format:
Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2023. How To Optimize My Blockchain? A Multi-Level Recommendation Approach. *Proc. ACM Manag. Data* 1, 1, Article 24 (May 2023), 27 pages. https://doi.org/10.1145/3588704

1 INTRODUCTION

When blockchains were first introduced, they supported only simple cryptocurrency exchange transactions [50]. However, over time blockchains evolved to support complex transactions using smart contracts, thus entering the arena of decentralized transactional management systems such as distributed databases [64]. Since blockchains target consensus in a trustless environment, they cannot easily match the performance of databases [9, 16, 22, 26, 53, 59, 80]. However, with the advent of permissioned blockchains that offer access control and transaction execution policies, blockchains strive to improve their performance while still providing at least partially decentralized trust [3, 5, 28, 48].

Apart from the proliferation of new blockchain system designs, there is highly vibrant and diverse ongoing research in the domain of system optimizations that focus on performance enhancements

*Work done while at Technical University of Munich

Authors' addresses: Jeeta Ann Chacko, chacko@in.tum.de, Technical University of Munich, Germany; Ruben Mayer, ruben.mayer@uni-bayreuth.de, University of Bayreuth, Germany; Hans-Arno Jacobsen, jacobsen@eecg.toronto.edu, University of Toronto, Canada.

2836-6573/2023/5-ART24 $15.00
https://doi.org/10.1145/3588704

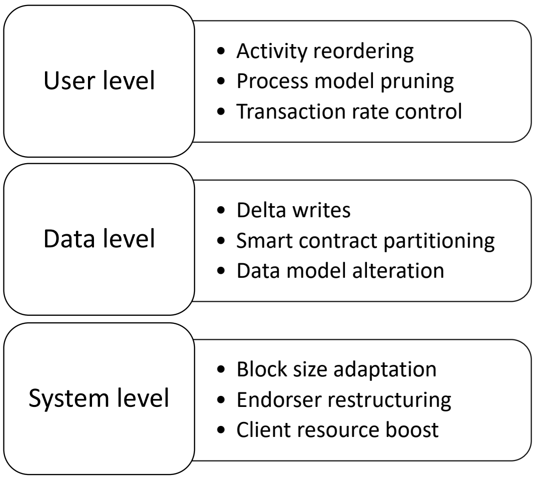


Fig. 1. Multi-level blockchain optimization

within prevailing permissioned blockchains [13, 27, 36, 37, 41, 54, 65–68]. The vast range of the literature targets control parameter tuning [13, 41, 68], transaction execution remodeling [27, 37, 66], and smart contract optimization [54]. However, we notice that a collective approach that encompasses all these optimization possibilities for a particular blockchain under the same umbrella is missing. Further, the literature falls short for an end-to-end optimization approach that includes not only system-level tuning and data remodeling but also process model redesign. Since permissioned blockchains are mainly employed by enterprises, a model-driven approach is often followed where the setup of the blockchain network, its transaction regulations, the underlying smart contract, and the data model are primarily based on a business process model created specifically for a particular application [21, 40, 56, 63, 69]. Such process models may be designed by business domain experts who are unaware of performance implications. For example, in Hyperledger Fabric (a.k.a. Fabric) [5], many transaction failures arise due to the order in which the transactions are executed [13, 65, 67]. Such failures could be reduced if the client processes that issue the transactions followed a different business logic in the first place. The prominence of data management while executing business processes has often been highlighted by the database community [11, 20, 34]. We make a similar argument for the importance of the process view in blockchains since the aspects covered by blockchains are manifold and not limited to data alone.

Therefore, given the numerous optimizations possible within a given blockchain system, their varying influence on a case-by-case basis [6, 13, 23, 51, 68, 81], and the resulting implementation efforts, there is a pressing need for a recommendation system that guides the user in selecting effective optimization strategies suitable for the blockchain under consideration depending on the specific use-case. Again, we can draw parallels from the exhaustive literature on parameter tuning and indexing recommendations for databases [1, 2, 42, 73]. However, since blockchains juggle multiple factors such as organizational governance [62], database definitions [59], consensus algorithms [46], provenance tracking [60], and smart contract design [47], a holistic perspective to optimization recommendations is desirable, which is currently lacking.

To address this gap, we propose a multi-level optimization recommendation approach for blockchains that provides to the users a comprehensive understanding of the different optimization possibilities for their blockchain system, thus enabling them to make a well-informed decision. Inspired by the abstraction levels in databases [45], we define three levels of abstraction for blockchain optimizations: system, data, and user-level (cf. Figure 1). The system-level recommendations include identifying ideal system configurations such as the block size or endorsement policy. The data-level recommendations deal with understanding the data model and optimizing smart contracts. The user-level recommendations focus on business process models and workloads induced by client processes. For example, we identified two activities in a digital rights management scenario (cf. Section 5.2) that frequently cause transaction conflicts and recommend a process model redesign to reduce such failures. Our approach can also verify compliance with the new process model. We design and implement a recommendation tool named BlockOptR that analyzes the blockchain logs from Fabric, one of the most widely used blockchains by enterprises [61], to demonstrate the performance improvements yielded by our approach.

Our contributions can be summarized as follows:

(1) We define a multi-level optimization recommendation approach that extensively analyzes the blockchain log and recommends optimization possibilities from different perspectives. Our method helps users gain a comprehensive understanding of their current system and make educated decisions regarding optimization strategies.

(2) We provide a formal definition for our recommendation strategies based on common attributes, such that any blockchain log with similar attributes can reuse our approach. We also discuss how our approach translates to different blockchain platforms, thereby providing the reader with a technology-independent outlook.

(3) We automate the extraction, preprocessing, and event log generation techniques for Fabric blockchain data. Thus, our tool BlockOptR will help to ease further research in the area of log-based analysis such as process mining in blockchains, since a preprocessed blockchain log can be directly obtained.

(4) We demonstrate the effectiveness of the optimization recommendations by implementing and evaluating them. Our experiments indicate an average of 20% improvement in the percentage of successful transactions and an average of 40% improvement in latency after applying the recommendations by BlockOptR.

(5) We extensively evaluate BlockOptR with three different types of workloads: A set of 24 synthetic workloads generated with a wide range of control variables, four widespread use case-based workloads from the literature, and a real-world event log of a loan application process. Thus, we cover a wide range of scenarios in our experimentation that are representative for real blockchain applications. This aids in overcoming the lack of publicly available data that restricts current research on process mining in permissioned blockchains. The BlockOptR tool, all the smart contracts, the workload generation scripts, and all the event logs are released as open-source to encourage further research in this area [10].

(6) We further establish the positive effect of our holistic recommendation approach on top of existing blockchain optimizations. Thus, we highlight that BlockOptR complements existing system-level blockchain optimization strategies such as FabricSharp [65] and Fabric++ [67] by adding higher-level optimizations.

2 BACKGROUND

2.1 Hyperledger Fabric

Fabric is an open-source permissioned blockchain system popularly used by enterprises [5]. The main components of Fabric are a smart contract (called chaincode), a distributed immutable ledger, a distributed world state database, a set of distributed peers, and an ordering service. The smart contract defines all the supported transactions on the blockchain. The transaction flow in Fabric follows an execute-order-validate (EOV) model [70]. The EOV model of Fabric is comparable to optimistic concurrency control in databases [31] and is therefore prone to multi-version concurrency control (MVCC) conflicts, which result in transaction failures.

(1) In the execution phase, transaction proposals are created by clients and sent to the endorsers. Endorsers are a set of specific peers that have the authority to execute the smart contract to endorse a transaction. An endorsement policy is configured to define the number of required endorsers for a transaction to be deemed valid. Endorsers generate read-write sets after smart contract execution. The transaction proposal and the read-write sets are signed by the endorsers and sent back to the clients.

(2) In the ordering phase, the clients forward these transactions to the ordering service. The ordering service orders the transactions into blocks using Raft [55], a crash fault-tolerant consensus algorithm, and sends them to all the peers in the network. Configurable parameters limit the number of transactions included in a block (block size) in terms of the number of transactions (block count), a timeout (block timeout), or the size of transactions in bytes (block bytes). Blocks are created whenever the buffered set of incoming transactions satisfies any of the three conditions.

(3) In the validation phase, every peer validates every transaction. Every peer in the Fabric network has a copy of the distributed ledger and the world state. Peers validate both the endorser signatures based on the endorsement policy and the read-write set with the current world state. If the validation is successful, the world state is updated. Else, a failure is detected. If the endorsement validation fails, it is called an endorsement policy failure; if the read-write set validation fails, it is called an MVCC read conflict. MVCC read conflicts for range reads are called phantom read conflicts. Regardless of the success or failure of the validation, all transactions are appended to the distributed ledger. Also, in the literature, MVCC read conflicts are often classified into inter-block and intra-block failures depending on whether the conflicting transactions reside in the same block or different blocks in the blockchain [13, 67].

2.2 Event Logs and Process Mining

An event log is a record of process executions over time. Process mining [75] is the technique of deriving a process model that exhibits the most frequent behaviors in an event log. It is mainly used for *process discovery* which helps to understand the underlying process model, *conformance checking* where deviations between a predicted process model and the actual behavior of the process can be identified and *model enhancement* where bottlenecks are identified and removed. The minimum data required in an event log for process mining are:

(1) `CaseID`: To distinguish different executions of the same process. Example: `ProductID` in a supply chain management related event log. A complete execution of a process is called a trace.
(2) `Activity name`: To identify the different steps in a process. Example: `Ship` or `Unload` activity in a supply chain management related event log.
(3) `Timestamp`: To determine the order of the different activities.

The event log can also have other attributes such as process owner, resources, and cost. Various algorithms are used to derive the process model such as alpha miner [76], heuristics miner [79] and fuzzy miner [30]. The core concept of all these algorithms is to analyze the different traces of the set of activities in the log and simplify the traces through abstraction or aggregation to produce a complete process model. Various open-source and commercial process mining tools are available (ProM [78], Disco [29], Celonis [12]).

3 A PROCESS PERSPECTIVE TO BLOCKCHAINS

Our work posits blockchain optimization as a holistic methodology rather than a pure system-level approach by introducing a process perspective. In this section, we emphasize the necessity and effectiveness of understanding the dependency between business processes and the performance of the blockchain through examples. Further, these examples motivate the need for an optimization recommender since many process-level optimizations can only be employed with approval from the decision-making bodies of an organization and, in most cases, cannot be automatically applied.

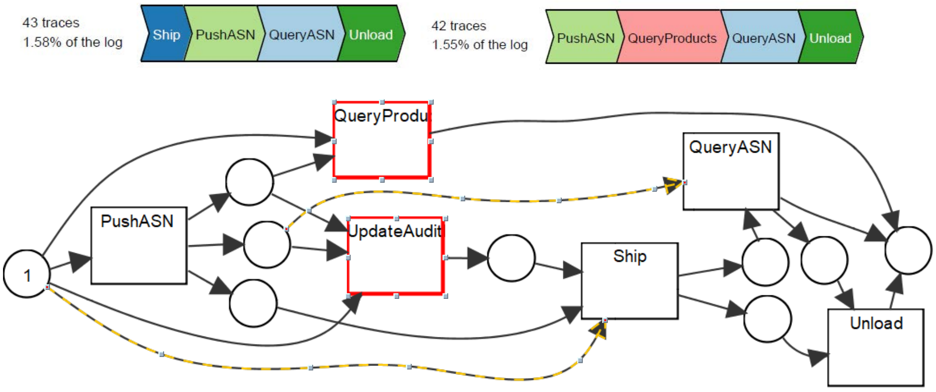
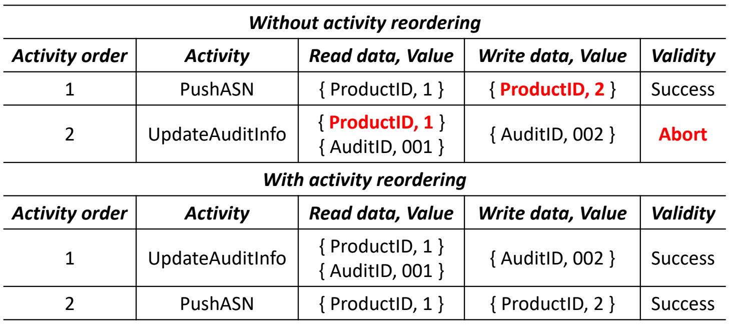


Fig. 2. Derived process model for SCM scenario

Process model pruning is an example of a process-level optimization that positively affects the system's performance. Figure 2 shows the process model derived from the blockchain log of a supply chain management (SCM) scenario. The highlighted paths and the traces embedded in the figure identify two unnecessary branches in the process model. Unless the advanced shipping notice is pushed (PushASN), one should never execute the Ship activity. Similarly, the Unload activity should never be executed unless a product has been shipped. Such illogical activity paths can arise due to manual errors or transaction failures, and the smart contract is designed to handle such issues, as we explain in the following example.

If the Unload transaction executes without a corresponding Ship, the transaction will only read the state but not modify it. However, it is up to the smart contract designer to either fail the transaction upon execution or commit the read-only transaction to the blockchain. Both these designs have their trade-offs. Committing the transaction adds an immutable record on the blockchain, which helps to track, for example, individuals or organizations who deviated from the expected process model. In a supply chain management scenario specifically, this is critical since the entire pipeline is distributed, and the primary purpose of the blockchain here is to provide data provenance among untrusted participants. However, on the other hand, failing a transaction immediately upon execution ensures that such unnecessary transactions do not go through all the time-consuming phases (ordering and validation), which can improve the system performance. We observe a 27% improvement in throughput and 19% increase in success rate of transactions when unnecessary activity paths are pruned in the smart contract (Section 6.2, Figure 13). The pruning can also be implemented at the process execution level by enforcing incentive or penalty measures for organizations or individuals that adhere to or deviate from the expected process model. This approach ensures that system performance is not prioritized over data provenance and hence, combines the advantages of both smart contract designs we discussed above.

Another cause of failures are transactional dependencies, and research in serialization algorithms has effectively reduced such failures through transaction reordering [65, 67]. However, reordering algorithms are expensive, as they basically need to solve the NP-hard problem of generating conflict-free dependency graphs [67]. An increase in endorsement policy failures due to inconsistent world



Without activity reordering				
Activity order	***Activity***	***Read data, Value***	***Write data, Value***	***Validity***
1	PushASN	{ ProductID, 1 }	{ **ProductID, 2** }	Success

2	UpdateAuditInfo	{ **ProductID, 1** }
		{ AuditID, 001 }
			{ AuditID, 002 }	**Abort**

With activity reordering				
Activity order	***Activity***	***Read data, Value***	***Write data, Value***	***Validity***

1	UpdateAuditInfo	{ ProductID, 1 }
		{ AuditID, 001 }
			{ AuditID, 002 }	Success

2	PushASN	{ ProductID, 1 }	{ ProductID, 2 }	Success

Fig. 3. Transaction dependency conflict example

states and the inability to handle large range queries are known problems of transaction reordering [13]. A different approach to the problem of dependency conflicts is to identify reorderable and unreorderable [65] *activities* instead of transactions. While the literature analyzes the keys accessed by transactions to understand serializability, the data model needs to be analyzed for process-level serialization. If two concurrent activities read the same data element but write to different elements in the data model then such activities are reorderable.

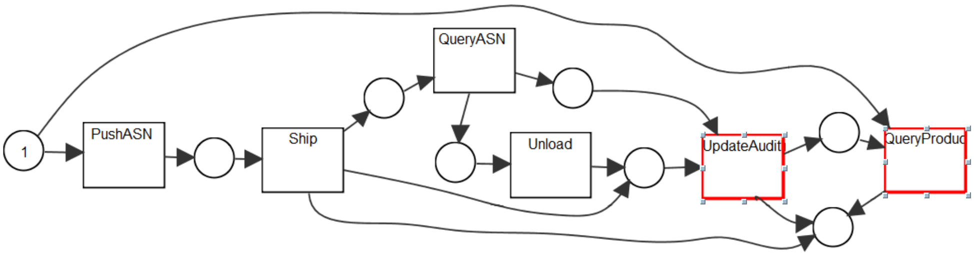


Fig. 4. Derived process model after activity reordering

For example, in the same supply chain management scenario, the UpdateAuditInfo activity reads a productID and writes an auditID, whereas the PushASN, Ship, and Unload activities read and write to the productID. Therefore, the pairs {UpdateAuditInfo, PushASN}, {UpdateAuditInfo, Ship} and {UpdateAuditInfo, Unload} are reorderable activities while {PushASN, Ship, Unload} are unreorderable. Figure 3 shows an example of a reorderable pair of activities where UpdateAuditInfo can succeed if it is executed either after the commit or before the execution of PushASN. Based on the business logic, it may be possible to impose procedures to restrict or reschedule certain activities to execute only at specific periods. For example, the corresponding process model in Figure 2 shows that UpdateAuditInfo occurs frequently between PushASN and Ship activities and therefore, UpdateAuditInfo may be executed before the transactions of the other two activities commit. However, UpdateAuditInfo is not a time-critical activity and can be rescheduled to take place only at specific times when traffic is low on the supply chain. We observe a 24% increase in throughput and 15% increase in success rate of transactions after a corresponding redesign where UpdateAuditInfo and QueryProducts activities are executed after PushASN, Ship, Unload. The new process model derived from the blockchain log confirms the adherence to the new design (Figure 4). Thus, by identifying conflicting activities, the process model can be redesigned to reduce transaction conflicts before they take place.

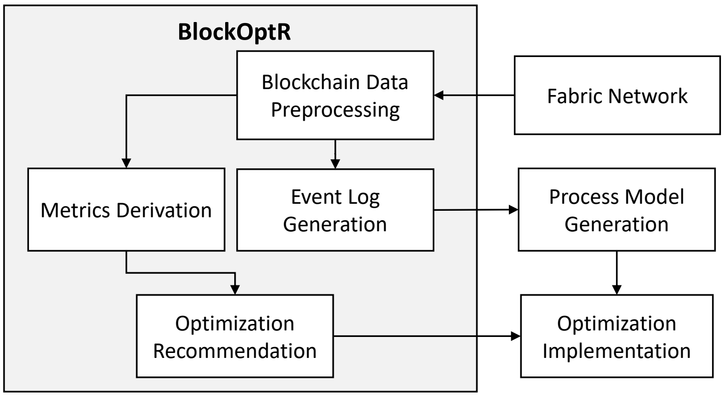


Fig. 5. BlockOptR workflow

4 BLOCKCHAIN OPTIMIZATION RECOMMENDER

We introduce an approach to recommend optimizations from three different abstraction levels: system, data, and user-level. The primary requirement to design and implement such a multi-level recommendation system is reliable data on all three levels. Knowledge about the system configurations (e.g., block size) and performance (e.g., throughput, transaction failures) is vital for generating system-level recommendations. Information about the current data model and access patterns, such as key distribution and dependencies, is essential for data-level recommendations. Lastly, knowledge concerning the use-case, business processes, and transaction workload is necessary for user-level recommendations. It is important to note that such information is not restricted to a specific level but is helpful across all levels. For example, the system-level performance can indicate the need for optimizations at all three levels.

The very definition of a blockchain implies the availability of a distributed ledger with immutable data regarding every transaction executed overtime. If we consider smart contracts, then every execution of the smart contract results in a transaction that is logged in the ledger. We consider this data (hereafter referred to as the blockchain log) as the primary source to derive optimization recommendations since, to our knowledge, such a distributed ledger consisting of all transactions is available for most blockchains. Therefore, our transaction-centric approach to deriving blockchain optimization recommendations is applicable to different blockchains.

We preprocess the raw data from the blockchain to create a blockchain log. Then, we obtain the values for key metrics which are used to detect multi-level optimization recommendations. Process mining strategies are then applied to the blockchain log to derive the process model. We identify the applicable optimizations using the recommendations and the derived process model. Figure 5 illustrates the workflow of our approach. We automated the main elements of this workflow as a tool, BlockOptR [10], implemented in Python and Node.js.

4.1 Blockchain Data Preprocessing

BlockOptR registers as a client on the Fabric network, reads the entire blockchain and saves it as JSON files. Next, the log is cleaned by removing the configuration and setup-related transactions that are not relevant and converted to CSV format. All information regarding each transaction executed in the Fabric network is logged on the blockchain. We extract seven attributes and derive two attributes from this extensive logged data. These attributes enable the derivation of multiple metrics required to recommend optimizations. The output of the data preprocessing step is a blockchain log with the following nine attributes.

(1) **Client timestamp**: The time at which the client generated the transaction.
(2) **Activity name**: The name of the smart contract function whose execution created the transaction. A(x) defines the activity name of a transaction x.
(3) **Function arguments**: The value of the parameters of the smart contract function.
(4) **Endorsers**: The set of all endorsers of the transaction.
(5) **Invokers**: The set of all clients and their respective organization who invoked the transactions.
(6) **Read-write set**: The set of keys accessed (read or written) by the transaction. The separate read set and write set of a transaction are also kept. RWS(x), RS(x) and WS(x) correspondingly define the read-write set, read set and write set of a transaction x.
(7) **Transaction status**: The status of the transaction that can have the values `success`, `MVCC read conflict` (MRC), `phantom read conflict` and `endorsement policy failure`. ST(x) defines the status of a transaction x.
(8) **Transaction type**: The type of transaction which is derived from the read-write set. This can have the values `read`, `write`, `update`, `range read` and `delete`. Transaction type is derived from the read-write set. TT(x) defines the type of a transaction x.
(9) **Commit order**: The order of the transactions in the blockchain log is the order in which transactions were committed to the blockchain.

4.2 Event Log Generation

The blockchain log extracted from the Fabric network can be used as an event log to apply process mining techniques that assist in recommending user-level optimizations. However, unlike the event logs created by process-aware information systems [74], a `CaseID` is not directly available in the event log extracted from a blockchain. Also, in most of the use-cases we observed, no single attribute is common to all activities that can be directly used as the `CaseID`. Therefore, we need to derive a common element for each use-case based on domain knowledge [4, 8, 17, 19, 44]. Since we are interested in a transactional perspective of the process model, we find a common element for all activities by analyzing the function arguments and read-write sets available in the log. For example, in the SCM scenario the productKey is a common element for all activities and is a suitable choice since the use-case is specifically related to tracking multiple products. This process of extracting the common element is automated for all the use-cases in this paper and can be easily extended for other use-cases. Once a common element is identified, we define a trace as a unique set of activities with the same value for the common element. We then assign a new `CaseID` to every trace.

Further, only the time at which the clients sent the transaction (client timestamp) is available in our log. However, there is no guarantee that the same order in which clients send their transactions will be maintained when the transactions are committed to the blockchain. Therefore, to derive the process model accurately, we use the commit order in place of the timestamp. Thus, with the generated `CaseID` and extracted/derived attributes, we have a complete event log. Now, any process mining technique can be applied to the event log to derive a process model. For example, we used the Alpha algorithm to derive the process models shown in Figure 2 and 4 [76].

4.3 Metrics

We define a set of metrics by scrutinizing multiple blockchain logs and analyzing metrics from the literature.

(1) **Rate metrics**: BlockOptR calculates the average transaction rate as well as the transaction rate distribution over time intervals from the event log. **Transaction rate (*Tr*)** is the average rate at which transactions are sent from the clients and is derived from the total transactions in the log and the client timestamps. **Transaction rate distribution (Trd_i)** is the transaction rate at a

specific interval i derived from the log. A user-configurable interval size (*ins*) in seconds is used to calculate this metric. *Usage:* Transaction rate is a useful metric to understand the performance. The rate distribution provides insights regarding periods of high or low traffic.

(2) **Failure metrics**: Similar to *Tr*, the **total failure rate (*TFr*)** as well as the rates of each type of failure (MVCC read conflicts, phantom read conflicts, endorsement policy failures) are derived from the log. The failure rate distribution (Frd_i) is calculated similar to *Trd*. *Usage:* Failure metrics help to detect times of high transaction failures. Optimizations such as transaction rate control can be applied based on the failure metrics.

(3) **Block size**: The user-configured **block count (B_{count})** and **block timeout ($B_{timeout}$)** are extracted from the log. The **average number of transactions in a block ($B_{sizeavg}$)** is also derived from the log. $B_{sizeavg}$ is equivalent to the average block size and can also be defined as $min\{B_{count}, Tr * B_{timeout}\}$. *Usage:* $B_{sizeavg}$ along with the rate metrics helps a user understand the effectiveness of their block size configurations. For example, if *Tr* is 500, B_{count} is 100, $B_{timeout}$ is 1 and $B_{sizeavg}$ is 100, then 100 transactions are packed into a block when 500 transactions are actually available every second. This means more blocks than necessary are being created which is inefficient, as block creation is expensive. Similarly, if *Tr* is 100, B_{count} is 500, $B_{timeout}$ is 2, and $B_{sizeavg}$ is 200, then blocks are created only every 2 seconds and transactions are queued up for a waiting period before being put into blocks. Both scenarios lead to performance degradation. So, based on the value of $B_{sizeavg}$, the user can update B_{count} and $B_{timeout}$ to efficiently handle the transaction rate.

(4) **Endorser significance (*EDsig*)** defines the number of transactions endorsed by each endorsing peer. *Usage:* This metric helps in identifying endorser bottlenecks. Suppose a limited number of endorsers always carry out the endorsements. In that case, the user can consider distributing the transactions more evenly among the endorsers or expanding the set of endorsers.

(5) **Invoker significance (*IVsig*)** defines the number of transactions invoked by each client. *Usage:* This metric helps to identify clients and the corresponding organizations that invoke a majority of the transactions. Client resource allocation decisions of such organizations can be made based on this metric.

(6) **Key frequency (*Kfreq*)** is defined as the number of failed transactions that access a specific key. **Key significance (*Ksig*)** is defined as the number of activities that access a specific key. *HK* defines the set of hotkeys that have high key frequency based on user-configurable thresholds. *Usage:* Identifying the hotkeys assists the users to identify optimization possibilities in their smart contracts, and key significance helps to detect the exact activities (that correspond to smart contract functions) that access the hotkeys. For example, if several functions access the same key, then the different functions could be separated into multiple smart contracts. Every smart contract executes on a different world state, thereby reducing failures (see example in Section 5).

(7) **Data-value correlation (*corDV*)** defines that two transactions are correlated if both access a same set of keys and one of them fails. *Usage:* Data-value correlation helps to identify transaction dependencies. Such dependent transactions are the root cause of MVCC read conflicts [13]. Various optimization strategies, such as process model redesign and transaction rate control, can be applied to these correlated transactions to mitigate failures.

(8) **Proximity correlation (*corP*)** defines the distance between two transactions that have a high data value correlation. For example, if $corP(x, y) == 1$ then transaction y happened immediately after x with no transactions in between. Further, we also derive the **activity-based proximity correlation (*corPA*)** which defines the distance between transactions of the same activity. *Usage:* Analyzing if the proximity correlation is "less than the block size" or "greater than the block size" can reveal useful insights regarding inter- and intra-block failures. If intra-block failures are very high, smaller block sizes can potentially reduce failures [13]. This metric also helps to

Table 1. Formalization of optimization recommendations

Recommendations	**Necessary conditions**
Activity reordering	$if\ corDV(x, y) == 1 \wedge WS(x) \cap WS(y) == \emptyset$
Process model pruning	$if\ A(x) = A(y) \wedge TT(x) \neq TT(y)$
Transaction rate control	$if\ (Trd_i \geq Rt_1) \wedge (Frd_i \geq Trd_i * Rt_2)$
Delta writes	$if\ corPA(x, y) == 1 \wedge ST(x) == MRC \wedge$ $\lvert WS(x)\rvert == \lvert WS(y)\rvert == 1 \wedge WS(x) \pm 1 == WS(y)$
Smart contract partitioning	$if\ Ksig(HK_i) > 1$
Data model alteration	$if\ (Ksig(HK_i) == 1) \vee (\lvert HK\rvert == 1)$
Block size adaptation	$if\ (Tr \geq B_{sizeavg} * Bt) \vee (Tr < B_{sizeavg} * Bt)$
Endorser restructuring	$if\ EDsig(e) > \lvert TX\rvert * Et$
Client resource boost	$if\ IVsig(c) > \lvert TX\rvert * It$

where $x, y \in TX, e \in E, c \in I, HK_i \in HK$
TX, E, I, HK are set of all transactions, endorsers, invokers and hotkeys
Rt_1, Rt_2, Bt, Et, It are user configurable thresholds

choose between inter- or intra-block transaction reordering strategies offered by different Fabric optimizations [65, 67].

4.4 Optimization Recommendations

We use a multi-level approach to utilize the defined attributes and metrics for recommending blockchain-specific optimization strategies. The optimization recommendation techniques explained in this section include configurable thresholds. We define appropriate default values for these thresholds based on our analysis of multiple logs, but the user can adapt these default values to fine-tune the detection strategies. The necessary condition to recommend each optimization strategy is formalized in Table 1.

4.4.1 User Level Recommendations.

At the user level, it is essential to focus on the actual workload of the running application. The rate and order in which the transactions are generated and committed to the blockchain has a vital impact on performance. We analyze the rate, dependencies, and type of the transactions to recommend optimizations at the user level.

(1) **Activity reordering**: Reorderable pairs of transactions can be identified by using the data value correlation and the read-write set. BlockOptR identifies the activities corresponding to such transaction pairs and recommends a process model redesign. The redesign should ensure that the identified activities are restructured to reduce conflicts (cf. Section 3).

(2) **Process model pruning**: If activities deviate from an expected behavior, then process model pruning is recommended. The transaction type of all transactions related to an activity is analyzed to identify anomalies. Comparing the traces in the event log and the derived process model with the identified anomalies helps to detect model pruning opportunities (cf. Section 3).

(3) **Transaction rate control**: BlockOptR evaluates the transaction rate distribution over time and identifies times when the rate is very high. It then checks the failure rates in the same time interval. If the failure rate is also very high, rate control is recommended. Two configurable thresholds are used to tune the tolerance level of transaction rate and failures.

4.4.2 Data Level Recommendations.

For data-level recommendations, we focus on identifying the specific areas in the data model that can be optimized by analyzing transaction failures, proximity correlation, read-write sets, and key significance. This aids the user in altering the smart contract and thereby the underlying data model to improve performance.

(4) **Delta writes**: Update transactions that only perform increment or decrement operations can be converted to delta-writes. Delta writes enable writing to multiple unique delta keys, which can be aggregated whenever the current value is required. Reading the key before each write is also not required. Thus, update transactions are converted to write-only transactions that write to unique keys. This helps to reduce transaction dependency-related failures. Delta writes are recommended when a single key is incremented or decremented by a failed transaction.

(5) **Smart contract partitioning**: A possibility to reduce transaction dependencies is to split a smart contract into multiple ones. Each smart contract will access separate world states, thereby avoiding conflicts. The functionality of the original smart contract will not change because it is possible to invoke functions between the two smart contracts if interaction is required.

For example, in a music rights management scenario, if a key `MusicID` is found to be hot and multiple functions such as `Play()` and `viewMetaData()` access this same key, then one can separate the functions into two different smart contracts. In other words, the underlying database is split into two by duplicating the primary key (`MusicID`) across both and having different secondary keys in each. The play count of `MusicID` is recorded in one and metadata is read from the other (cf. Section 6.2). This is analogous to designing the table layout in relational databases. The smart contract needs to be analyzed and updated to implement this optimization. Smart contract partitioning is recommended if multiple activities access a hotkey.

(6) **Data model alteration**: If activities have a dependency on themselves, then a data model alteration can be beneficial to reduce transaction conflicts. For example, in a digital voting scenario, if a key `ElectionID` is found to be hot and is only accessed by the function `Vote()`, then a possible optimization is to use another primary key such as `VoterID`. Then, instead of updating all the votes together, the votes can be updated per voter (cf. Section 6.2). Further, if only a single hotkey is detected then it is beneficial to analyze the data model to understand the reason for the skewed access to this specific data element (cf. Section 6.3). Data model alteration is recommended if a hotkey is accessed only by a single activity or if a single hotkey is detected.

4.4.3 *System Level Recommendations*.

At the system level, we focus on two crucial system configuration settings that can significantly affect the performance of Fabric: the endorsement policy and the block size [13, 68]. Further, we also identify client bottlenecks to aid in resource allocation decisions. We use the endorser significance, invoker significance, transaction rate, and actual block size metrics to derive system-level optimization recommendations. Since these recommendations are based on the blockchain log generated by the running application, it helps the user to identify ideal configuration settings based on their current use-case and workload, leading to performance improvements.

(7) **Block size adaptation**: The average transaction rate (Tr), the average block size ($B_{sizeavg}$) and a configurable threshold (Bt) are used to recommend block size adaptation. The literature recommends smaller block sizes when transaction rates are lower and larger block sizes when the rates are higher [13, 68]. If the block size is too small, too many blocks are created, and block creation becomes a bottleneck. If the block size is too large, the block creation is delayed by waiting for sufficient transactions. Therefore, if block size adaptation is recommended, then set $B_{timeout}$ and B_{count} such that $min\{B_{count}, Tr * B_{timeout}\}$ is equal to Tr. We do not provide recommendations for *block bytes* adaptation since it is difficult to accurately derive the size of a transaction (that can include the transaction payload, endorser identities and other metadata) from the log.

(8) **Endorser restructuring**: For every Fabric transaction generated by the clients, the corresponding smart contract function is executed by the endorsers defined in the endorsement policy. Smart contract execution is a time and resource-consuming action. If the same endorsers receive a higher

load of transactions while others remain idle, this indicates a bottleneck or load imbalance. Such load imbalances can occur when the endorsement policy explicitly defines an endorsement as mandatory from a specific set of endorsers. For example, the endorsement policy `And(Org1,OR(Org2,Org3))` implies that an endorsement from `Org1` is mandatory. As a consequence, `Org1` could become an endorsement bottleneck. We detect endorser bottlenecks by identifying endorsers that endorse more transactions than a user-specified threshold. The default threshold values detect whether all the endorsers participate equally in the endorsement process. The threshold values can be adapted to increase or decrease the sensitivity to imbalances.

(9) **Client resource boost**: Multiple time-consuming tasks are performed by the clients in a Fabric network, including but not limited to transaction proposal invocation, endorser response verification, packing of endorser responses as a transaction, transaction submission to the ordering service, and collection of peer commit responses. The invoker significance metrics identify the clients and the corresponding organizations that invoke a majority of the transactions. This identification can assist in resource allocation decisions, such as increasing the number and size of clients registered to the identified organization. It could also point to problems in the underlying business process.

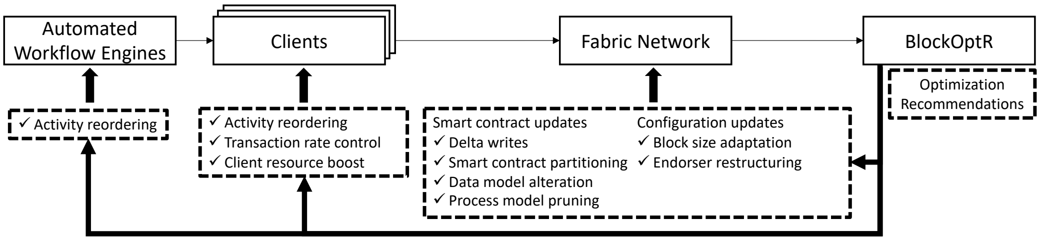


Fig. 6. Optimization implementation on a live Fabric network

4.5 Implementation of Optimizations

The recommended optimizations can be implemented in several ways. Figure 6 visualizes where the different recommendations can be implemented on a live Fabric network. Here, we show an automated workflow engine that triggers transactions based on a process model. These transactions are sent via the clients to the Fabric network. The logs of the Fabric network are analyzed by BlockOptR to generate optimization recommendations. Each of the recommended optimizations can be implemented at different levels as shown in the figure.

Activity reordering can be implemented by modifying the underlying process model in the workflow engine such that activities follow a conflict-free order. Alternatively, one can monitor the transactions on the clients and reorder either per client or across all clients using a client manager. Process model pruning can be implemented via organizational measures such as incentives or penalties to ensure that activities adhere to their expected behavior (not shown in the figure). However, pruning can also be implemented directly in the smart contract by early aborting anomalous transactions during the endorsement phase. Transaction rate control can be implemented in multiple ways. Each client can monitor their own transaction rate and perform load shedding or queuing. The same can be done across clients using a central monitor. A third approach is to monitor the transaction rate in the ordering service and apply load shedding there. Smart contract revisions are required to implement all the data-level optimizations. In Fabric, smart contract upgrades are possible on the fly without restarting the system [72]. Block size can be adapted either by changing the configuration file or by using a *configuration update* transaction in Fabric [71]. Endorser restructuring can be implemented by altering the endorsement policy. The policy can be changed in the Fabric configuration file or using a *configuration update* transaction [71]. Based on the transaction load per client identified by BlockOptR, client resources can be scaled if the current allocation appears insufficient to handle the load and the new clients can be dynamically registered to the Fabric network.

Our implementations. Although all optimizations can be applied in a live system on the fly, since our evaluation runs in an experimental environment, we restart the Fabric network after every experiment. We use the Caliper benchmarking system [35] which has a client manager that can be configured to order the transactions across clients and control the rate of transactions generated, thus emulating activity reordering and transaction rate control. The number of clients can also be scaled to demonstrate a client resource boost. Process model pruning and all data-level optimizations are implemented by analyzing and modifying the smart contract. Block size and endorsement policies are updated in the Fabric configuration file.

5 EXPERIMENTAL METHODOLOGY

We used version 2.0 of HyperledgerLab [13], which is an automated testbed for Hyperledger Fabric 2.2 integrated with the Caliper benchmarking system. We set up a Kubernetes cluster of 1 master and 5 worker nodes over which all the Fabric network components as well as Caliper components are distributed as Kubernetes pods. Each node runs on a Ubuntu Focal (20.04) virtual machine with 4 vCPUs and 9.8 GB RAM. We use 10 Caliper workers for our experiments. For every experiment, we measure the success rate which is the percentage of successful transactions out of the total number of transactions, the average latency and the throughput of all successful transactions.

5.1 Workload Generation

The content of the distributed ledger, which is used as the input to our tool, is a direct result of the workload executed on the blockchain. Therefore, we extensively evaluate BlockOptR by using three different types of workload. Also, after implementing the recommendations generated by BlockOptR, we rerun the experiments with the same workloads to analyze the effect of the optimization.

Table 2. Control variables

Control Variable	**Values (Default in bold)**
Workload type	**Uniform**, Read-heavy, Insert-heavy, Update-heavy, RangeRead-heavy
Endorsement policy	P1, P2, **P3**, P4
Endorser distribution skew	**0**, 6
Key distribution skew	**1**, 2
Number of organizations	**2**, 4
Block count	50, **300**, 1000
Send rate	50, **300**, 1000
Transaction dist skew	**0**, 70%

5.1.1 *Synthetic workloads.*

We use an extended version of a synthetic workload generator that can generate synthetic workloads based on a set of control variables for a generic smart contract *genChain* [13]. We use a range of values for these control variables described in Table 2 to generate multiple workloads of 10,000 transactions each. The endorsement policies used in our experiments are:
P1: `And(Org1, Or(Org2,Org3,Org4))`
P2: `And(Or(Org1,Org2), Or(Org3,Org4))`
P3: `Majority(Org1,...,OrgN)`
P4: `OutOf(2,Org1,Org2,Org3,Org4)`

Table 3. Settings to implement optimization

Optimizations Recommended	Settings
Activity reordering	Reorder workload generation
Transaction rate control	Set send rate to 100 TPS
Process model pruning	Update smart contract
Delta writes	
Smart contract partitioning	
Data model alteration	
Block size adaptation	Set block count to derived transaction rate
Endorser restructuring	Set endorsement policy to P4
Client resource boost	Double clients for recommended organization

By generating synthetic workloads, we ensure that multiple realistic scenarios are covered in our experiments. We then evaluate BlockOptR with each of these workloads to generate optimization recommendations. Further, we implement each of the recommended optimizations to evaluate the performance improvement.

5.1.2 *Use-case based workloads.*

Secondly, we use extended versions of four popular use-case based smart contracts from the literature [13] and generate workloads. BlockOptR is then used to generate optimization recommendations with these workloads. The four smart contracts we use are as follows.

Supply Chain Management (SCM): This smart contract defines the operations of a logistics network that includes sending an advanced shipping notice, shipping a product, reading the shipping notice and unloading the product (in this order). There is also a query operation to query the information of the different products (`queryProducts`) and a `updateAuditInfo` function that updates an audit entry with the product details. These can happen at any point in time. We generated a workload of 10,000 transactions based on these assumptions by sending in order the transactions `pushASN`, `ship`, `queryASN` and `unload` while the transactions `queryProducts` and `updateAuditInfo` are sent randomly.

Digital Rights Management (DRM): This smart contract manages the rights of artists in the music industry. The smart contract includes a *Play* function that is executed whenever a piece of music is played by any user. The other smart contract functions include adding a new piece of music, querying the rights, viewing the metadata and calculating the revenue of the right holders. In a realistic scenario, the *Play* transaction would be executed far more frequently than all the other transactions. Therefore, we create a *Play* heavy workload for this use-case. We generate 10,000 transactions randomly where 70% of the transactions are *Play*. The remaining 30% comprise all the other transactions generated uniformly at random.

Electronic Health Records (EHR): In this smart contract, patients can provide or revoke access rights to medical institutes as well as research institutes to query their medical records. We assume that the number of patients would be more than the other participants and generate a 70% update-heavy workload of 10,000 transactions.

Digital Voting (DV): This smart contract includes a function to vote in an election, query the parties, query the results as well as end the election. We can assume that during an actual election there will be periods of high traffic while the voting is taking place. Therefore, we generate a workload which initially sends 1,000 `queryParties` transactions at a rate of 100 TPS, then 5,000 `Vote` transactions at a rate of 300 TPS and finally 1 `seeResults` and `endElection` transaction each.

5.1.3 *Loan Application Process (LAP).*

Table 4. Experiments with the synthetic workload

Experiment Number	Control variable	Value	Optimizations recommended
1	Endorsement Policy	P1	Endorser restructuring Activity reordering
2	Endorsement Policy / Endorser dist skew	P2 / 6	Endorser restructuring Activity reordering
3	No: of orgs	4	Transaction rate control
4	Workload	Read-heavy	Activity reordering
5	Workload	Update-heavy	Transaction rate control
6	Workload	Insert-heavy	Activity reordering
7	Workload	RangeRead-heavy	Activity reordering
8	Key distribution skew	2	Transaction rate control Activity reordering Smart contract partitioning
9	Block count	50	Block size adaptation Activity reordering
10	Block count	300	Transaction rate control Activity reordering
11	Block count	1000	Transaction rate control Activity reordering
12	Send rate	50	Activity reordering
13	Send rate	300	Activity reordering
14	Send rate	1000	Block size adaptation Transaction rate control Activity reordering
15	Transaction distribution skew	70%	Transaction rate control Activity reordering Client resource boost

Thirdly, we created a smart contract and workload using a real-life event log of the loan application process of a Dutch financial institute which is available publicly [77] together with the corresponding process flow [57]. We extracted all the events of the first 2,000 loan applications and created 20,000 corresponding transactions. We then created a smart contract where every activity in the loan application process flow has a corresponding smart contract function. The event log contains an `employeeID` for every employee in the bank handling loan applications and an `applicationID` for every loan application processed by the bank. The smart contract we implemented uses the `employeeID` as the key and the value of the key is an array of structures where every structure includes the `applicationID`, `loan type`, `loan amount` and `loan status`. Therefore, querying a specific `employeeID` will easily provide all the applications processed by that employee. We then executed the 20,000 transactions on the smart contract at a low rate of 10 TPS to simulate a real world scenario where manually processing the loan applications takes a long time. We also ran the same experiment at a higher rate of 300 TPS to emulate an automated loan application and validation process. We use BlockOptR to generate optimization recommendations which help to improve the smart contract implementation and thereby the performance.

Though the LAP event logs are from a database setting, this is a realistic use-case for blockchains as an automated loan application system requires security and decentralized trust (e.g., micro-loans, decentralized loan applications, and more generally DeFi [33, 58, 82, 83]). Consequently, this experiment demonstrates the utility of BlockOptR in a realistic scenario. In the use-case based experiments, all the transactions followed the expected order based on the assumptions we defined. In contrast, with this real event log, we evaluate the real order in which the transactions are executed which can deviate from the process model.

6 EXPERIMENTAL RESULTS

We exhaustively evaluate our recommendation approach with a wide range of workloads and smart contracts. Please note that, whenever transaction rate control is implemented there is an expected decrease in the throughput. However, clients benefit heavily from higher success rates, and the apparent decrease in the throughput is just closer to the sustainable throughput of the system. In all our experiments the default value for the thresholds are $Et = 0.5$, $Rt1 = 300$, $Rt2 = 0.3$, $Bt = 0.6$ and $It = 0.5$. All the settings including the control variable values changed to implement each recommended optimization is shown in Table 3.

6.1 Synthetic Workloads

Due to space restrictions, we present 15 workloads in Table 4. The full list of experiments and results can be seen in our repository [10]. The control variable that is tuned for each experiment is shown along with its value. All the other control variables have the default value shown in Table 2. Experiments 1 to 15 are conducted with no optimizations applied and then BlockOptR is used to derive optimization recommendations. The recommendations generated by BlockOptR are also shown in Table 4. Since the synthetic smart contract has a simple logic with no branches, increment/decrement operations or complex data model, process model pruning, delta writes and data model alterations are not recommended here. Next, we implement the recommended optimizations and re-execute all the experiments. The results of the experiments are grouped based on the optimization recommendations and can be seen in Figures 7, 8, 9, 10, 11 and 12. We also explain how the thresholds are set for our experiments and how they can be tuned by users.

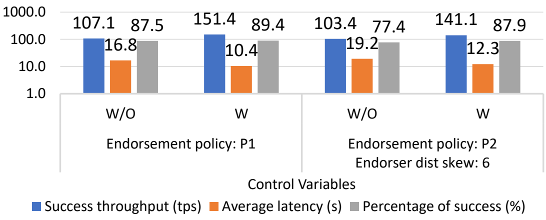


Fig. 7. Endorser restructuring

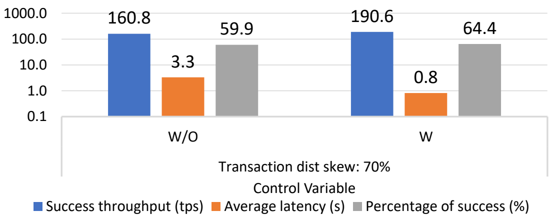


Fig. 8. Client resource boost

6.1.1 ***Endorser restructuring.*** : The effect of endorser restructuring can be seen in Figure 7. When the endorsement policy is P1, all the clients must send their transactions to Org1 due to the specific endorsement policy and hence, an endorsement bottleneck is detected for `Org1`. Since the endorsement policy requires signatures from two organizations, we change the policy to `OutOf(2,Org1,Org2,Org3,Org4)` so that the clients can distribute the transactions evenly among all endorsers. This optimization leads to a 29% increase in throughput (Figure 7). In Experiment 2, since the endorser distribution is skewed, the clients send transactions unevenly and therefore two of the organizations endorse far more often than the other two. We re-executed the experiment with an even distribution of transactions to the endorsers and observe a 26% increase in throughput (Figure 7). The main impact of this optimization is on throughput and latency as it reduces transaction queuing on few specific peers and instead distributes them evenly.

We set the thresholds for this recommendation such that we expect an even distribution of transactions to all endorsers, i.e., even minor bottlenecks are detected. This can be tuned to detect only severe bottlenecks. Further, since these are synthetic experiments, changing the endorsement policy is not critical. In real scenarios, consultation with the governing bodies of an enterprise is required before changing the policy. Still, the recommendations by BlockOptR help to highlight bottlenecks which in turn can convince the management to change the policy.

6.1.2 ***Client resource boost***. : Figure 8 shows the effect of client resource scaling. After increasing the number of clients, we observe a 75% decrease in latency, a 15% increase in throughput, and a 7% increase in success rate. The thresholds are set such that this optimization is recommended when more than 50% of transactions are invoked by the same organization. This can be fine-tuned to detect less severe bottlenecks.

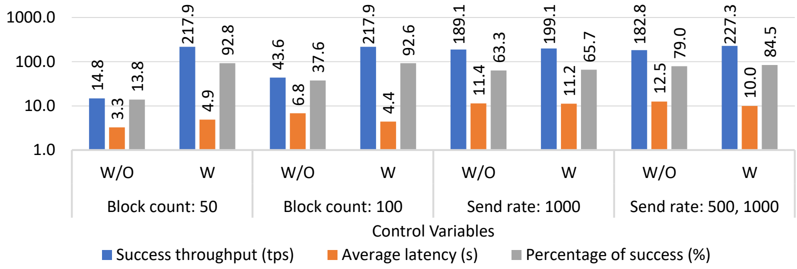


Fig. 9. Block size adaptation

6.1.3 ***Block size adaptation***. : The effect of block size adaptation can be seen in Figure 9. In our experiments, we use the default block time out of 1s. Therefore, we make the block count equal to the transaction rate whenever the block size adaptation is recommended. After changing the block size, we observe up to 93% improvement in throughput and 85% improvement in success rate (Figure 9; Block count: 50). The thresholds are set such that this optimization is recommended whenever the average block size is 60% larger or smaller than the transaction send rate derived from the log. The thresholds can be decreased to make the recommendation more sensitive to transaction rate changes.

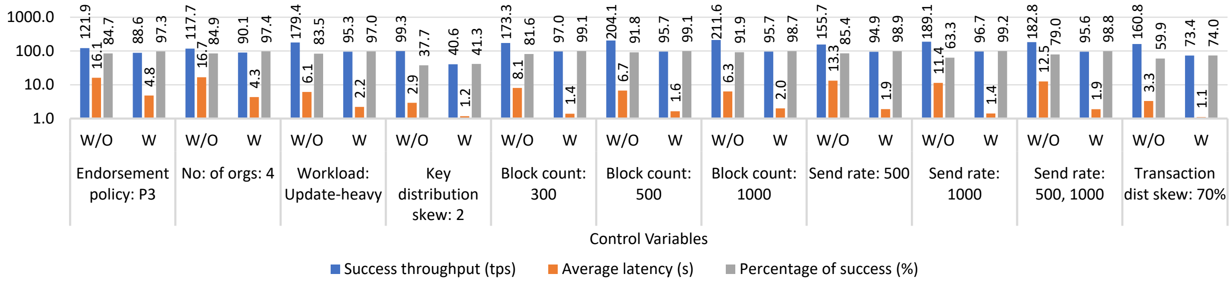


Fig. 10. Transaction rate control

6.1.4 ***Transaction rate control***. : The effect of transaction rate control is shown in Figure 10. In these experiments, periods of high traffic (around 300 TPS) were also identified as periods of high failure rates. We then lowered the transaction send rate to 100 TPS on the clients and re-executed the experiments. We observe significant improvement of up to 87% in latency and 36% in success rate (Figure 10; Send rate: 1000). We set the thresholds for this recommendation at 300 TPS which is the default send rate of our experiments. This means that we consider the current traffic of the system as high and want to detect periods of failure. Users can adjust this threshold based on what is considered high (more than the sustainable traffic rate) for their Fabric network installation.

6.1.5 ***Activity reordering***. : The effect of activity reordering can be seen in Figure 11. We observe that BlockOptR recommends activity reordering for all experiments except Experiments 3 and 5 (Table 4). Reordering was suggested for two activities (`Read` and `Update`) which conflict with each other. We updated the configuration of the client manager to generate read transactions before

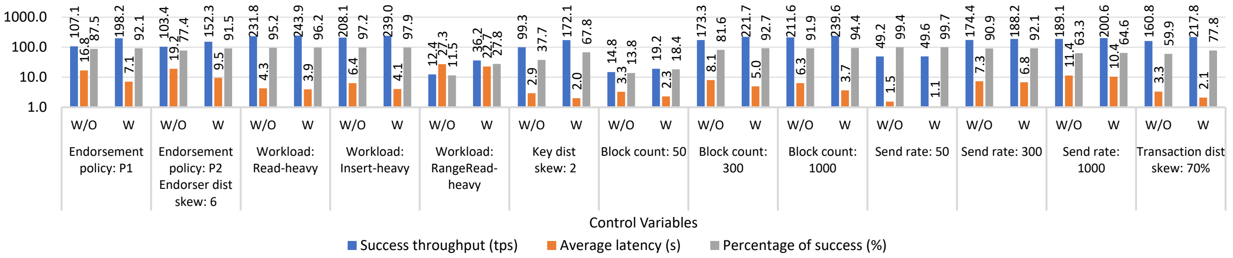


Fig. 11. Activity reordering

all other transactions. This implementation emulates a scenario where organizational measures were applied to enforce activity reordering. We then re-executed the experiments and observe a performance improvement in all the experiments. There is up to 65% increase in throughput and 58% increase in success rate (Figure 11; Workload: RangeReadheavy). We have set the thresholds such that if 40% of the MVCC failures are caused by activities that can be reordered, this strategy is recommended. This can be made more lenient by increasing the threshold such that reordering is suggested only in very significant cases. For Experiments 3 and 5, less than 40% of MVCC conflicts are caused by the two activities where reordering is possible. For example, the activity `Update` has a dependency on itself which cannot be removed by reordering.

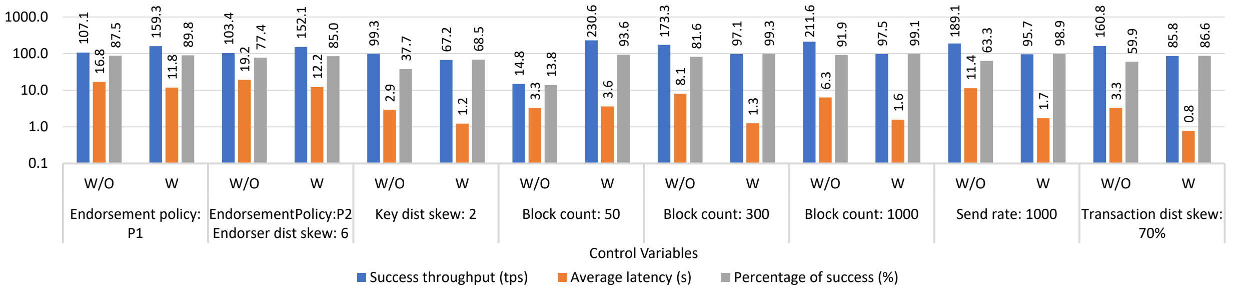


Fig. 12. All recommended optimizations combined

6.1.6 *Combined optimizations.*

: We also executed the experiments after applying all the recommended optimizations together. We observe up to a 93% improvement in throughput and 85% improvement in the success rate (Figure 12: Block count: 50). In all the experiments, the performance obtained by applying all the optimizations is comparable to the performance yielded by the optimization with the highest improvement.

Further remarks. Though smart contract partitioning is recommended for Experiment 8, this optimization requires understanding the functionality of the smart contract. Unfortunately, for the synthetically generated smart contract that includes only generic read, update and insert functions, we cannot redesign the smart contract.

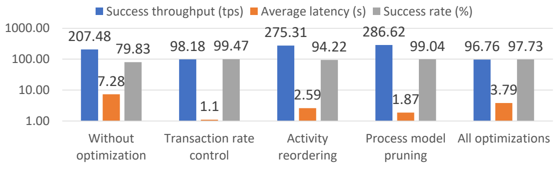


Fig. 13. SCM use-case

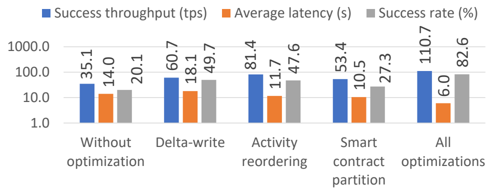


Fig. 14. DRM use-case

6.2 Use-case based Workloads

Supply Chain Management (SCM): With the SCM use-case, three optimizations are recommended by BlockOptR: activity reordering, process model pruning and transaction rate control (Figure 13). After implementing reordering for the reorderable activities (queryProducts and UpdateAuditInfo), we observe a 24% increase in throughput and 15% increase in success rate. Pruning was recommended for the Ship activities that occur without or before the PushASN activity. It was also recommended to prune Unload activities that occur without or before the Ship activity. We adapted the smart contract to implement the pruning recommendation. This resulted in a 27% improvement in throughput and 19% increase in success rate. Transaction rate control and applying all recommendations together also improves the performance.

Digital Rights Management (DRM): With the DRM use-case, three optimizations are recommended by BlockOptR: activity reordering, delta-writes and smart contract partitioning. Figure 14 shows the results of applying these optimizations. To implement the delta write recommendation, we observed that the Play function in the smart contract has an increment operation to count the number of times a piece of music was played. We converted this into a delta write and the delta-keys are aggregated whenever the calcRevenue function is invoked (since it requires the play count). With this optimization, we can observe a significant improvement of 42% in throughput and 50% in success rate. However, the average latency increases in this case because the calcRevenue function now takes up more time for aggregation. Since calcRevenue is not executed as frequently as Play, the overall performance is not affected though.

Activity reordering was recommended for calcRevenue and queryRightHolders functions and we reconfigured the clients to send these activities after all other activities. This emulates a scenario where an organization restricts specific transactions to specific time periods. We observe more than 50% increase in both throughput and success rate with this optimization.

Hot keys were detected and frequently used by four activities. We analysed the smart contract and discovered that, though all four functions have a dependency on the same key, the functionalities are different. Play and calcRevenue need only the play count, while viewMetaData and queryRightHolders need metadata and not the play count of a piece of music. Therefore, we split the smart contract into two, where one smart contract has the Play and calcRevenue functions and the second smart contract has the other two functions. The create function is included in both smart contracts, and invocation of the first smart contract invokes the same function in the second smart contract. We observe a 35% increase in throughput and a 26% increase in success rate with this optimization. Applying all the optimizations together improves the performance by more than 50%.

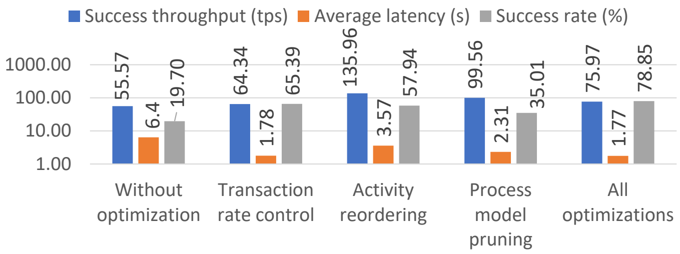


Fig. 15. EHR use-case

Electronic Health Records (EHR): In this use-case, three optimizations were recommended: activity reordering, process model pruning and transaction rate control (Figure 15). Activity reordering for the read activities resulted in a 60-65% improvement in throughput and success rate. When the smart contract was updated to prune illogical paths (revoke access to records without granting access), we observe around 43% increase in throughput and success rate. After applying transaction rate control, a 69% increase in success rate was observed. All optimizations applied together also improve the performance.

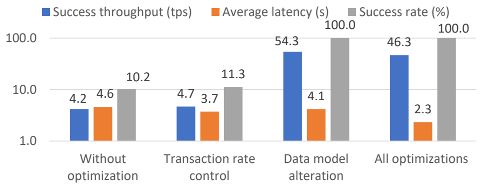


Fig. 16. Digital voting use-case

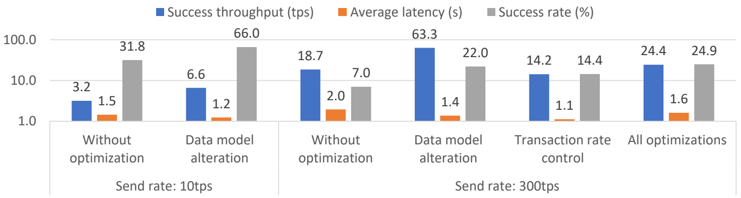


Fig. 17. Loan application process use-case

Digital Voting (DV): In this use-case, two optimizations were recommended: transaction rate control and data model alteration. The results are shown in Figure 16. High failure rates were detected for periods when the `Vote` transactions were frequent. After applying transaction rate control, a slight improvement of 11% in throughput was observed. The hotkeys were detected and most frequently used by the `Vote` function resulting in a recommendation to alter the data model. We analysed the smart contract and observed that `partyID` was used as the key for the vote function which is invoked by multiple voters during the voting phase. We redesigned the smart contract such that `voterID` is assigned as the primary key. Since voters are restricted to a single vote, we observe 100% success rate with this new smart contract because there are no more transaction dependencies. We also observe an improvement in the performance when both optimizations are applied together.

6.3 Loan Application Process (LAP)

The optimization recommended for the LAP use-case was data model alteration (Figure 17). The `employeeID` 1 had a high key frequency since this employee processed the highest number of loan applications. We then re-implemented our smart contract and assigned `applicationID` as the key and modeled the value as a structure that includes `employeeID`, `loan amount`, `loan type` and `loan status`. This new implementation helped to remove the hot key and yielded more than 50% improvement in throughput and success rate for both the lower and higher send rates.

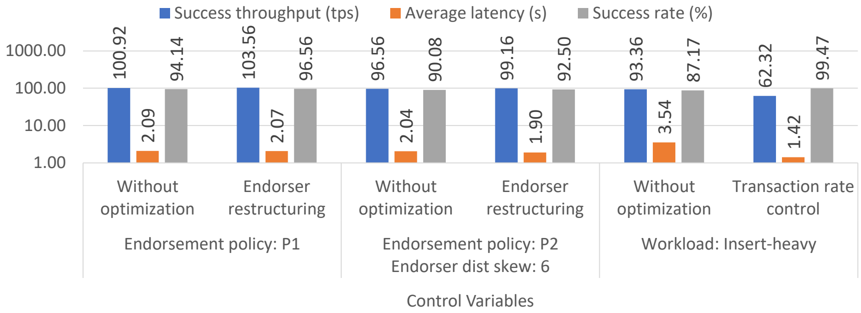


Fig. 18. Synthetic workloads with FabricSharp

6.4 Fabric Extensions

As a holistic recommendation approach, our work lies orthogonal to existing Fabric optimizations in the literature. In this section, we demonstrate how our approach works on top of two optimized

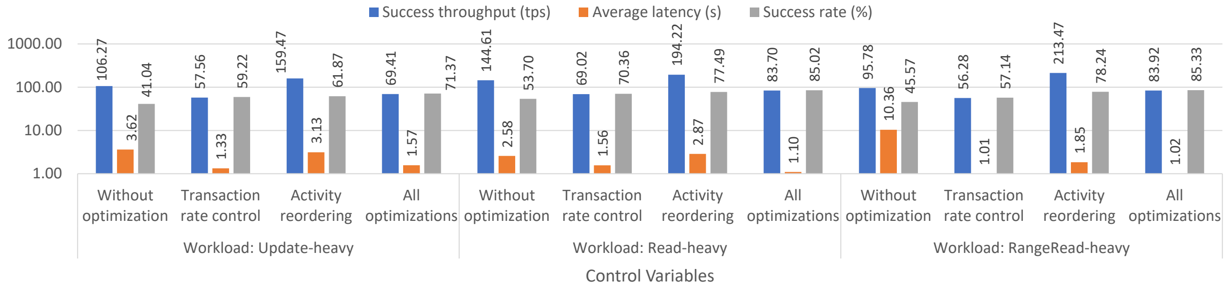


Fig. 19. Synthetic workloads with Fabric++

extensions of Fabric: FabricSharp [65] and Fabric++ [67]. Both implement different transaction reordering strategies that mitigate MVCC read conflicts. The Fabric++ scheduler is integrated in the FabricSharp implementation [25] and we use this for our experiments. We executed the synthetic workloads on both and then used BlockOptR to generate recommendations. The literature says FabricSharp increases endorsement policy failures and is less performant for insert-heavy workloads while Fabric++ is least performant with an update-heavy, read-heavy and range-read-heavy workloads [13]. Therefore, we execute these specific experiments shown in Figures 18 and 19 with the synthetic workloads. Activity reordering, transaction rate control and endorser restructuring were recommended and by implementing these recommendations, we observe up to a 55% increase in throughput and 46% increase in success rate (Figure 19: RangeRead-heavy workload). Our experiments with these Fabric extensions show that even with effective system-level optimizations, Fabric can still benefit from optimizations at all levels of abstraction.

7 LESSONS LEARNED AND LIMITATIONS

We demonstrated that BlockOptR is capable of effectively recommending suitable optimization strategies. Further, we also explained how to implement these optimizations and quantified the performance improvements after implementation. This section discusses the insights we gained from our experiments.

User level optimizations. Activity reordering was one of the most frequently recommended optimizations in our experiments. We highlight use-cases such as SCM where such reordering can be applicable. Our model pruning recommendation emphasizes that identifying incompetencies in the process model can lead not only to efficient process execution but also improve the performance of the underlying system. Load shedding or queuing is often employed when systems cannot handle the workload. Using our recommendations, specific activities and time periods can be identified where such rate control techniques are most effective. For example, rate control is recommended for the `Vote` activity in the digital voting use-case. Therefore, instead of system-wide rate control, only the specific clients that deal with the identified activities need to employ rate control techniques.

Data level optimizations. These optimizations show how the design of the smart contract and the data model significantly influence the performance. The smart contract is initially designed with a specific process model in mind. However, we understand how the smart contract is being used in practice by analyzing the blockchain logs. BlockOptR pinpoints functions and keys that cause bottlenecks which in turn helps the smart contract developer to make appropriate modifications.

System level optimizations. Setting the endorsement policy is a management decision that often excludes discussions with the technical team designing the blockchain. Our recommendations highlight the need to bring together management and technical discussions to decide optimal configuration settings. Further, we also demonstrate the need to verify whether the policy is being used effectively. For example, even if the policy defines the equal distribution of endorsements, the clients may send their transactions in a skewed manner. In such instances, we recommend enforcing a management measure, such as dividing the endorsers equally among the clients such that clients of one organization only send transactions to specific endorsers. The compliance with such measures can also be checked by BlockOptR. Block size optimization is frequently discussed in the literature and associated with the transaction rate of a system [13, 36, 68]. Instead of system-level changes such as using transaction rate monitors, we derive the transaction rate and the actual block size from the log. This helps to understand traffic patterns over time and find reasonable block size settings. While the literature mainly focuses on optimizing the peers and ordering service components of Fabric [27, 65, 67], our client-related recommendations highlight the need to focus on client-side optimizations as well.

Technology Independence. Our multi-level recommendation approach is demonstrated using the Fabric blockchain. Technology independence is difficult to attain due to the vast implementation variations between the numerous blockchain systems and the corresponding differences in the contents of the distributed ledger. However, we draw attention to specific examples which can guide future researchers to translate our approach to other blockchain systems. In Quorum, the block time or mining frequency has a linearly proportional influence on the transaction latencies [7] which is analogous to our block size adaptation recommendation strategy. Also, Corda has the concept of notaries to attest transactions where distributing the transactions over multiple notaries is expected to improve the throughput [18]. This is again comparable to our endorsement restructuring recommendation. Further, there are numerous gas-fee reduction and vulnerability detection strategies for Ethereum smart contracts in the literature [54] which translate to our recommendations at the data level. Tools like Lorikeet and Caterpiller automate the conversion and execution of process models as Ethereum smart contracts, which would make it easier to implement the user-level optimizations that we recommend [43, 69].

Limitations. The optimizations recommended by BlockOptR need to be manually implemented by the user. A self-adaptive system with a feedback loop that automatically implements the recommendations is possible. However, in an enterprise scenario, for many of the optimizations such as endorser restructuring, activity reordering, and process model pruning, management level approvals might be required before implementation. Additionally, for applications that do not follow a specific process model, the event logs can be misleading. In such scenarios, user-level optimizations such as activity reordering and process model pruning are not relevant. Therefore, domain knowledge about the use-case is required for implementing the recommended optimizations appropriately. Further, our implementation of some of the optimizations such as transaction rate control are trivial in such benchmarking scenarios and do not account for real-world overheads. However, the implementations are mainly for demonstrative purposes. Our work focuses on the multi-level recommendation approach used by BlockOptR rather than the implementation of the optimizations. Finally, our experiments without and with the recommended optimizations are done on similar workloads generated with the same input parameters, i.e., we assume a continued trend in the pattern of the workload after the optimizations are applied. However, in scenarios where the workload fluctuates or the optimization implementation is delayed, BlockOptR may need to be re-executed to generate new recommendations.

8 RELATED WORK

The literature proposes various Fabric optimization strategies such as transaction reordering [13, 65, 67], block size optimizations [13, 36], CRDTs [52], and parallelizing various components [27]. Our work lies orthogonal to such optimization strategies and focuses on an optimization recommendation approach. We demonstrate how our recommendations can be used along with two of the literature's optimization strategies to improve performance further.

There is also extensive research in the database community on index and query optimizations that include self-tuning systems as well as recommendation systems [2, 14, 15, 38, 73]. Though we can draw parallels from these research, our work focuses on blockchain-specific optimization recommendations. Different configuration settings (such as block size and endorsement policy) and the concept of smart contracts introduce new dimensions to the recommendation approach, which are not required for databases.

There is ongoing research on applying process mining techniques on blockchains to derive process-level insights [24, 32, 39, 49]. Klinkmüller et al. [39] and Mühlberger et al. [49] describe different approaches to extract process data from the Ethereum blockchain. Hobeck et al. [32] use process mining on an Ethereum-based betting application to identify shortcomings in the application. Process mining on blockchains currently only focuses on permissionless blockchains as they are publicly accessible. However, deriving and studying the process model is equally critical for private blockchains, and therefore, our work contributes to this less explored area of research. Further, unlike the related work, we focus on using process mining for recommending blockchain optimization strategies. We only found a single paper that uses permissioned blockchains, where Duchmann et al. [24] extract process data from Fabric and detect semantic errors in a smart contract. Though our work is comparable, we extract not only the process data but also blockchain-specific attributes from Fabric, derive multiple metrics, and recommend optimization strategies.

There is extensive research in the database community in the domain of data-aware business processes that encourage a business process perspective to database management systems [11, 20, 34]. Calvanese et al. [11] comprehensively survey the contributions in this realm and catalog contributions from various fields, including database theory and process management. These works were an important motivation for us to view blockchains from a business process perspective. However, our work brings new contributions since blockchains deal with several other elements apart from data, such as smart contracts and endorsement policies.

9 CONCLUSIONS

This paper showcases the necessity and effectiveness of having a holistic perspective on blockchain optimizations. We define a multi-level recommendation approach based on several metrics and attributes derived from the blockchain log. We define a total of nine optimizations at the system, data, and user-level of a blockchain. We implement an automated optimization recommendation tool, BlockOptR, based on these concepts. Further, we demonstrate how such optimizations can be implemented to improve the system performance. After implementing the recommended optimizations, we observe an average of 20% improvement in the success rate and an average of 40% improvement in latency. We extensively evaluate the system with a wide range of workloads covering multiple real-world scenarios. We hope to inspire enterprises to use our contributions to detect blockchain optimization strategies and to contribute their live blockchain (anonymized) logs for further research in this domain. The BlockOptR tool, all the smart contracts, the workload generation scripts, and all the event logs are available as open-source [10]. We also plan to extend our tool to include more optimization recommendations.

In terms of future work, we are currently developing a ProM plugin which would provide a user-friendly interface for BlockOptR. Presently, the threshold settings of BlockOptR depend on the business network setup. For example, the rate threshold for our setup was 300 TPS as higher rates led to instabilities, but this can vary for other deployments. Therefore, tuning these thresholds automatically in BlockOptR could be a future extension. Another interesting extension is to define additional attributes that applications can log, thereby providing more data for optimization recommendations. Further, investigating the effect of workload fluctuations and delay in applying the recommendations is another challenging future direction.

ACKNOWLEDGMENTS

This work is funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 392214008, and by the Bavarian Cooperative Research Program of the Free State of Bavaria - DIK-2002-0013//DIK0114/02.

REFERENCES

[1] Parinaz Ameri. 2016. Challenges of index recommendation for databases: With specific evaluation on a NoSQL database. In *dalam 28th GI-Workshop on Foundations of Databases (Grundlagen von Datenbaken), Nörten-Hardenberg, Germany.*

[2] Parinaz Ameri. 2016. On a self-tuning index recommendation approach for databases. In *2016 IEEE 32nd International Conference on Data Engineering Workshops (ICDEW).* 201–205. https://doi.org/10.1109/ICDEW.2016.7495648

[3] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. *Permissioned Blockchains: Properties, Techniques and Applications.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3448016.3457539

[4] Analyzing the complaints process at Granada city council 2020. https://www.tf-pm.org/resources/casestudy/analyzing-the-complains-prociess-at-granada-city-council.pdf. [Online; accessed 12-April-2023].

[5] 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. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18).* ACM, New York, NY, USA, Article 30, 15 pages. https://doi.org/10.1145/3190508.3190538

[6] Arati Baliga, Nitesh Solanki, Shubham Verekar, Amol Pednekar, Pandurang Kamat, and Siddhartha Chatterjee. 2018. Performance Characterization of Hyperledger Fabric. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT).* 65–74. https://doi.org/10.1109/CVCBT.2018.00013

[7] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. 2018. Performance Evaluation of the Quorum Blockchain Platform. https://doi.org/10.48550/ARXIV.1809.03421

[8] Dina Bayomie, Iman Helal, Ahmed Awad, Ehab Ezat, and Ali Elbastawissi. 2015. Deducing Case IDs for Unlabeled Event Logs, Vol. 256. https://doi.org/10.1007/978-3-319-42887-1_20

[9] Sara Bergman, Mikael Asplund, and Simin Nadjm-Tehrani. 2020. Permissioned blockchains and distributed databases: A performance study. *Concurrency and Computation: Practice and Experience* 32, 12 (2020), e5227. https://doi.org/10.1002/cpe.5227 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5227 e5227 cpe.5227.

[10] BlockOptR 2022. https://github.com/jeetachacko/BlockOptR. [Online; accessed 12-April-2023].

[11] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. 2013. Foundations of Data-Aware Process Analysis: A Database Theory Perspective. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (New York, New York, USA) *(PODS '13).* Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2463664.2467796

[12] Celonis Process Mining 2022. https://www.celonis.com/. [Online; accessed 12-April-2023].

[13] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21).* Association for Computing Machinery, New York, NY, USA, 221–234. https://doi.org/10.1145/3448016.3452823

[14] Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. 2009. Query Recommendations for Interactive Database Exploration. In *Scientific and Statistical Database Management*, Marianne Winslett (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–18.

[15] Surajit Chaudhuri and Vivek Narasayya. 2007. Self-Tuning Database Systems: A Decade of Progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria) *(VLDB '07).* VLDB Endowment, 3–14.

[16] Mohammad Jabed Morshed Chowdhury, Alan Colman, Muhammad Ashad Kabir, Jun Han, and Paul Sarda. 2018. Blockchain Versus Database: A Critical Analysis. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE).* 1348–1353. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00186

[17] Combining Multiple Columns as Case ID 2020. https://fluxicon.com/book/read/perspectives/#combining-multiple-columns-as-case-id. [Online; accessed 12-April-2023].

[18] Corda 2022. https://docs.r3.com/en/platform/corda/4.10/enterprise/key-concepts-notaries.html. [Online; accessed 12-April-2023].

[19] Data Requirements: Case ID 2020. https://fluxicon.com/book/read/dataext/#case-id. [Online; accessed 12-April-2023].

[20] Daniel Deutch and Tova Milo. 2011. A Quest for Beauty and Wealth (or, Business Processes for Database Researchers). In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Athens,

Greece) *(PODS '11)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/1989284.1989286

[21] Claudio Di Ciccio, Alessio Cecconi, Marlon Dumas, Luciano García-Bañuelos, Orlenys López-Pintado, Qinghua Lu, Jan Mendling, Alexander Ponomarev, An Binh Tran, and Ingo Weber. 2019. Blockchain support for collaborative business processes. *Informatik Spektrum* 42, 3 (2019), 182–190.

[22] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. 2018. Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Transactions on Knowledge and Data Engineering* 30, 7 (2018), 1366–1385. https://doi.org/10.1109/TKDE.2017.2781227

[23] Julian Dreyer, Marten Fischer, and Ralf Tönjes. 2020. Performance Analysis of Hyperledger Fabric 2.0 Blockchain Platform. In *Proceedings of the Workshop on Cloud Continuum Services for Smart IoT Systems* (Virtual Event, Japan) *(CCIoT '20)*. Association for Computing Machinery, New York, NY, USA, 32–38. https://doi.org/10.1145/3417310.3431398

[24] Frank Duchmann and Agnes Koschmider. 2019. Validation of smart contracts using process mining. In *ZEUS. CEUR workshop proceedings*, Vol. 2339. 13–16.

[25] FabricSharp Git Repository 2022. https://github.com/ooibc88/FabricSharp. [Online; accessed 12-April-2023].

[26] Ghareeb Falazi, Vikas Khinchi, Uwe Breitenbücher, and Frank Leymann. 2019. Transactional properties of permissioned blockchains. *SICS Software-Intensive Cyber-Physical Systems* (2019). https://doi.org/10.1007/s00450-019-00411-y

[27] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 455–463. https://doi.org/10.1109/BLOC.2019.8751452

[28] Gideon Greenspan et al. 2015. Multichain private blockchain-white paper. *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf* (2015), 57–60.

[29] Christian W Günther and Anne Rozinat. 2012. Disco: Discover Your Processes. *BPM (Demos)* 940 (2012), 40–44.

[30] Christian W Günther and Wil MP Van Der Aalst. 2007. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*. Springer, 328–343.

[31] Theo Härder. 1984. Observations on optimistic concurrency control schemes. *Information Systems* 9, 2 (1984), 111 – 120. https://doi.org/10.1016/0306-4379(84)90020-6

[32] Richard Hobeck, Christopher Klinkmüller, Hmn Dilum Bandara, Ingo Weber, and Wil Van der Aalst. 2021. *Process Mining on Blockchain Data: a Case Study of Augur.* Technical Report. EasyChair.

[33] Christian Hugo Hoffmann. 2021. Blockchain Use Cases Revisited: Micro-Lending Solutions for Retail Banking and Financial Inclusion. *Journal of Systems Science and Information* 9, 1 (2021), 1–15. https://doi.org/doi:10.21078/JSSI-2021-001-15

[34] Richard Hull. 2008. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In *On the Move to Meaningful Internet Systems: OTM 2008*, Robert Meersman and Zahir Tari (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1152–1163.

[35] Hyperledger Caliper 2020. https://hyperledger.github.io/caliper/. [Online; accessed 12-April-2023].

[36] Zsolt István, Alessandro Sorniotti, and Marko Vukolić. 2018. Streamchain: Do blockchains need blocks?. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. 1–6.

[37] Haris Javaid, Chengchen Hu, and Gordon Brebner. 2019. Optimizing Validation Phase of Hyperledger Fabric. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 269–275. https://doi.org/10.1109/MASCOTS.2019.00038

[38] Sadhana J. Kamatkar, Ajit Kamble, Amelec Viloria, Lissette Hernández-Fernandez, and Ernesto García Cali. 2018. Database Performance Tuning and Query Optimization. In *Data Mining and Big Data*, Ying Tan, Yuhui Shi, and Qirong Tang (Eds.). Springer International Publishing, Cham, 3–11.

[39] Christopher Klinkmüller, Alexander Ponomarev, An Binh Tran, Ingo Weber, and Wil van der Aalst. 2019. Mining blockchain processes: Extracting process mining data from blockchain applications. In *International Conference on Business Process Management*. Springer, 71–86.

[40] Olga Labazova, Erol Kazan, Tobias Dehling, Tuure Tuunanen, and Ali Sunyaev. 2021. Managing Blockchain Systems and Applications: A Process Model for Blockchain Configurations. *arXiv preprint arXiv:2105.02118* (2021).

[41] Mengting Liu, F. Richard Yu, Yinglei Teng, Victor C. M. Leung, and Mei Song. 2019. Performance Optimization for Blockchain-Enabled Industrial Internet of Things (IIoT) Systems: A Deep Reinforcement Learning Approach. *IEEE Transactions on Industrial Informatics* 15, 6 (2019), 3559–3570. https://doi.org/10.1109/TII.2019.2897805

[42] Jiaheng Lu, Yuxing Chen, Herodotos Herodotou, and Shivnath Babu. 2019. Speedup Your Analytics: Automatic Parameter Tuning for Databases and Big Data Systems. *Proc. VLDB Endow.* 12, 12 (aug 2019). https://doi.org/10.14778/3352063.3352112

[43] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alex Ponomarev. 2018. CATERPILLAR: A Business Process Execution Engine on the Ethereum Blockchain. https://doi.org/10.48550/ARXIV.1808.03517

[44] Heidy M. Marin-Castro and Edgar Tello-Leal. 2021. Event Log Preprocessing for Process Mining: A Review. *Applied Sciences* 11, 22 (2021). https://doi.org/10.3390/app112210556

[45] Dennis McLeod and John Miles Smith. 1980. Abstraction in Databases. In *Proceedings of the 1980 Workshop on Data Abstraction, Databases and Conceptual Modeling* (Pingree Park, Colorado, USA). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/800227.806871

[46] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. 2017. A review on consensus algorithm of blockchain. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2567–2572. https://doi.org/10.1109/SMC.2017.8123011

[47] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. 2018. An Overview of Smart Contract and Use Cases in Blockchain Technology. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 1–4. https://doi.org/10.1109/ICCCNT.2018.8494045

[48] JP Morgan. 2016. Quorum whitepaper. *New York: JP Morgan Chase* (2016).

[49] Roman Mühlberger, Stefan Bachhofner, Claudio Di Ciccio, Luciano García-Bañuelos, and Orlenys López-Pintado. 2019. Extracting event logs for process mining from data stored on the blockchain. In *International Conference on Business Process Management*. Springer, 690–703.

[50] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[51] Q. Nasir, Ilham A. Qasse, M. Talib, and A. B. Nassif. 2018. Performance Analysis of Hyperledger Fabric Platforms. *Secur. Commun. Networks* 2018 (2018), 3976093:1–3976093:14.

[52] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. 2019. FabricCRDT: A Conflict-Free Replicated Datatypes Approach to Permissioned Blockchains. In *Proceedings of the 20th International Middleware Conference* (Davis, CA, USA) *(Middleware '19)*. Association for Computing Machinery, New York, NY, USA, 110—122. https://doi.org/10.1145/3361525.3361540

[53] Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. 2019. Blockchain Meets Database: Design and Implementation of a Blockchain Relational Database. *Proc. VLDB Endow.* 12, 11 (jul 2019). https://doi.org/10.14778/3342263.3342632

[54] Keerthi Nelaturu, Sidi Mohamed Beillahi, Fan Long, and Andreas Veneris. 2021. Smart Contracts Refinement for Gas Optimization. In *2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*. 229–236. https://doi.org/10.1109/BRAINS52497.2021.9569819

[55] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Philadelphia, PA) *(USENIX ATC'14)*. USENIX Association, Berkeley, CA, USA, 305–320. http://dl.acm.org/citation.cfm?id=2643634.2643666

[56] Orlenys Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. 2019. Caterpillar: A business process execution engine on the Ethereum blockchain. *Software: Practice and Experience* (05 2019). https://doi.org/10.1002/spe.2702

[57] Process mining on the loan application process of a Dutch Financial Institute 2017. https://www.win.tue.nl/bpi/2017/bpi2017_winner_professional.pdf. [Online; accessed 12-April-2023].

[58] Yuncheng Qiao, Chaoqun Ma, Qiujun Lan, and Zhongding Zhou. 2019/12. Inventory Financing Model Based on Blockchain Technology. In *Proceedings of the Fourth International Conference on Economic and Business Management (FEBM 2019)*. Atlantis Press, 337–342. https://doi.org/10.2991/febm-19.2019.7

[59] Mayank Raikwar, Danilo Gligoroski, and Goran Velinov. 2020. Trends in Development of Databases and Blockchain. In *2020 Seventh International Conference on Software Defined Systems (SDS)*. 177–182. https://doi.org/10.1109/SDS49854.2020.9143893

[60] Aravind Ramachandran and Dr. Murat Kantarcioglu. 2017. Using Blockchain and smart contracts for secure data provenance management. *arXiv* (2017).

[61] Michel Rauchs, Apolline Blandin, Keith Bear, and Stephen B McKeon. 2019. 2nd global enterprise blockchain benchmarking study. *Available at SSRN 3461765* (2019).

[62] Olivier Rikken, Marijn Janssen, and Zenlin Kwee. 2019. Governance challenges of blockchain and decentralized autonomous organizations. *Information Polity* 24 (11 2019), 1–21. https://doi.org/10.3233/IP-190154

[63] Henrique Rocha and Stéphane Ducasse. 2018. Preliminary Steps Towards Modeling Blockchain Oriented Software. In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. 52–57.

[64] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. 2021. *Blockchains vs. Distributed Databases: Dichotomy and Fusion.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3448016.3452789

[65] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A Transactional Perspective on Execute-Order-Validate Blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New

York, NY, USA, 543–557. https://doi.org/10.1145/3318464.3389693

[66] Gary Shapiro, Christopher Natoli, and Vincent Gramoli. 2020. The Performance of Byzantine Fault Tolerant Blockchains. In *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*. 1–8. https://doi.org/10.1109/NCA51143.2020.9306742

[67] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the Lines Between Blockchains and Database Systems: The Case of Hyperledger Fabric. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. ACM, New York, NY, USA, 105–122. https://doi.org/10.1145/3299869.3319883

[68] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. 2018. 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)*. 264–276. https://doi.org/10.1109/MASCOTS.2018.00034

[69] An Binh Tran, Qinghua Lu, and Ingo Weber. 2018. Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management. In *BPM (Dissertation/Demos/Industry) (CEUR Workshop Proceedings, Vol. 2196)*. CEUR-WS.org.

[70] Transaction Flow 2022. https://hyperledger-fabric.readthedocs.io/en/release-2.2/txflow.html. [Online; accessed 12-April-2023].

[71] Updating a channel configuration 2022. https://hyperledger-fabric.readthedocs.io/en/release-2.2/config_update.html. [Online; accessed 12-April-2023].

[72] Upgrading a smart contract 2022. https://hyperledger-fabric.readthedocs.io/en/release-2.2/deploy_chaincode.html#upgrading-a-smart-contract. [Online; accessed 12-April-2023].

[73] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy Lohman, and Alan Skelley. 2000. DB2 advisor: an optimizer smart enough to recommend its own indexes. In *Proceedings of 16th International Conference on Data Engineering*. https://doi.org/10.1109/ICDE.2000.839397

[74] Wil van der Aalst. 2009. Process-Aware Information Systems: Lessons to Be Learned from Process Mining. *T. Petri Nets and Other Models of Concurrency* 2 (01 2009), 1–26. https://doi.org/10.1007/978-3-642-00899-3_1

[75] Wil Van Der Aalst. 2012. Process mining. *Commun. ACM* 55, 8 (2012), 76–83.

[76] Wil van der Aalst, T. Weijters, and L. Maruster. 2004. Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering* 16, 9 (2004), 1128–1142. https://doi.org/10.1109/TKDE.2004.47

[77] Boudewijn F van Dongen. 2017. https://doi.org/10.4121/12705737.v2

[78] Boudewijn F Van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP van Der Aalst. 2005. The ProM framework: A new era in process mining tool support. In *International conference on application and theory of petri nets*. Springer, 444–454.

[79] A.J.M.M. Weijters, Wil M.P. Aalst, van der, and A.K. Alves De Medeiros. 2006. *Process mining with the HeuristicsMiner algorithm*. Technische Universiteit Eindhoven.

[80] Karl Wüst and Arthur Gervais. 2018. Do you Need a Blockchain?. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 45–54. https://doi.org/10.1109/CVCBT.2018.00011

[81] Xiaoqiong Xu, Gang Sun, Long Luo, Huilong Cao, Hongfang Yu, and Athanasios V. Vasilakos. 2021. Latency performance modeling and analysis for hyperledger fabric blockchain network. *Information Processing & Management* 58, 1 (2021), 102436. https://doi.org/10.1016/j.ipm.2020.102436

[82] Qi Yang, Xiao Zeng, Yu Zhang, and Wei Hu. 2019. New Loan System Based on Smart Contract *(BSCI '19)*. Association for Computing Machinery, New York, NY, USA, 121–126. https://doi.org/10.1145/3327960.3332395

[83] Dirk A Zetzsche, Douglas W Arner, and Ross P Buckley. 2020. Decentralized Finance. *Journal of Financial Regulation* 6, 2 (09 2020), 172–203. https://doi.org/10.1093/jfr/fjaa010 arXiv:https://academic.oup.com/jfr/article-pdf/6/2/172/37064506/fjaa010.pdf

Received April 2022; revised July 2022; accepted August 2022

ACM License and Audio/Video Release

Title of the Work: How To Optimize My Blockchain? A Multi-Level Recommendation Approach

Author/Presenter(s): Jeeta Ann Chacko (Technical University of Munich);Ruben Mayer (Technical University of Munich, University of Bayreuth);Hans-Arno Jacobsen (University of Toronto)

Type of material:full paper

Publication: Proceedings of the ACM on Management of Data

1. Glossary

2. Grant of Rights

(a) Owner hereby grants to ACM an exclusive, worldwide, royalty-free, perpetual, irrevocable, transferable and sublicenseable license to publish, reproduce and distribute all or any part of the Work in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library, and to authorize third parties to do the same.

(b) In connection with software and "Artistic Images and "Auxiliary Materials, Owner grants ACM non-exclusive permission to publish, reproduce and distribute in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library.

(c) In connection with any "Minor Revision", that is, a derivative work containing less than twenty-five percent (25%) of new substantive material, Owner hereby grants to ACM all rights in the Minor Revision that Owner grants to ACM with respect to the Work, and all terms of this Agreement shall apply to the Minor Revision.
(d) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

☑ A. Grant of Rights. I grant the rights and agree to the terms described above.

□ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government?
○ Yes ● No
Country:

---

3. Reserved Rights and Permitted Uses.

(a) All rights and permissions the author has not granted to ACM in Paragraph 2 are reserved to the Owner, including without limitation the ownership of the copyright of the Work and all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM in Paragraph 2(a), Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Authors home page, (2) the Owner's institutional repository, or (3) any repository legally mandated by an agency funding the research on which the Work is based.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use; (viii) Bundle the Work in any of Owner's software distributions; and

(xi) Use any Auxiliary Material independent from the Work.

The rights management and bibstrip text blocks below will be added to the lower left hand portion of the first page of your published paper. As this text will provide rights information for your paper, please make sure that this text is displayed and positioned correctly when you receive your author proofs for review.

Authors should understand that consistent with ACM's policy of encouraging dissemination of information, each work published by ACM appears with a copyright and the following notice:

If you are using Authorized ACM TeX templates, the following code will generate the proper statements based on your rights choices. Please copy and paste it into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.

```
\setcopyright{acmlicensed}
\acmJournal{PACMMOD}
\acmYear{2023} \acmVolume{1} \acmNumber{1} \acmArticle{24}
\acmMonth{5} \acmPrice{15.00}\acmDOI{10.1145/3588704}
```

If you are using Word, copy and paste these words in the space provided at the bottom of your first page:

> Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
>
> 2836-6573/2023/MonthOfPublication - ArticleNumber $15.00
> © Copyright is held by the owner/author(s). Publication rights licensed to ACM.
> https://doi.org/10.1145/3588704

NOTE: DOIs will be registered and become active shortly after publication in the ACM Digital Library

---

4. ACM Citation and Digital Object Identifier.

(a) In connection with any use by the Owner of the Definitive Version, Owner shall include the ACM citation and ACM Digital Object Identifier (DOI).
(b) In connection with any use by the Owner of the Submitted Version (if accepted) or the Accepted Version or a Minor Revision, Owner shall use best efforts to display the ACM citation, along with a statement substantially similar to the following:

> "© [Owner] [Year]. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in {Source Publication}, https://doi.org/10.1145/{number}."

5. Audio/Video Recordiong of Presentation

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately by itself as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

A. Do you agree to the above Audio/Video Release? ● Yes ○ No

6. Auxiliary Materials, not integral to the Work

* Your Auxiliary Materials Release is conditional upon you agreeing to the terms set out below.

[Defined as additional files, including software and executables that are not submitted for review and publication as an integral part of the Work but are supplied by the author as useful resources for the reader.]

I hereby grant ACM permission to serve files containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that any of my Auxiliary Materials do not knowingly and surreptitiously incorporate malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software.

□ I agree to the above Auxiliary Materials permission statement.

□ This software is knowingly designed to illustrate technique(s) intended to defeat a system's security. The code has been explicitly documented to state this fact.

7. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

● We/I have not used third-party material.
○ We/I have used third-party materials and have necessary permissions.

8. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part IV and be sure to include a notice of copyright with each such image in the paper.

● We/I do not have any artistic images.
○ We/I have have any artistic images.

9. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants.

10. Enforcement.

At ACM's expense, ACM shall have the right (but not the obligation) to defend and enforce the rights granted to ACM hereunder, including in connection with any instances of plagiarism brought to the attention of ACM. Owner shall notify ACM in writing as promptly as practicable upon becoming aware that any third party is infringing upon the rights granted to ACM, and shall reasonably cooperate with ACM in its defense or enforcement.

11. Governing Law

This Agreement shall be governed by, and construed in accordance with, the laws of the state of New York applicable to contracts entered into and to be fully performed therein.

Funding Agents

1. Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) 392214008
2. Bavarian Cooperative Research Program of the Free State of Bavaria DIK-2002-0013//DIK0114/02

DATE: **03/24/2023** sent to chacko@in.tum.de at **09:03:04**

Appendix C

Should my Blockchain Learn to Drive? A Case of Hyperledger Fabric

Should my Blockchain Learn to Drive? A Study of Hyperledger Fabric

Jeeta Ann Chacko
chacko@in.tum.de
Technical University of Munich

Ruben Mayer
ruben.mayer@uni-bayreuth.de
University of Bayreuth

Hans-Arno Jacobsen
jacobsen@eecg.toronto.edu
University of Toronto

Abstract

Similar to other transaction processing frameworks, blockchain systems need to be dynamically reconfigured to adapt to varying workloads and changes in network conditions. However, achieving optimal reconfiguration is particularly challenging due to the complexity of the blockchain stack, which has diverse configurable parameters. This paper explores the concept of self-driving blockchains, which have the potential to predict workload changes and reconfigure themselves for optimal performance without human intervention. We compare and contrast our discussions with existing research on databases and highlight aspects unique to blockchains. We identify specific parameters and components in Hyperledger Fabric, a popular permissioned blockchain system, that are suitable for autonomous adaptation and offer potential solutions for the challenges involved. Further, we implement three demonstrative locally autonomous systems, each targeting a different layer of the blockchain stack, and conduct experiments to understand the feasibility of our findings. Our experiments indicate up to 11% improvement in success throughput and a 30% decrease in latency, making this a significant step towards implementing a fully autonomous blockchain system in the future.

1 Introduction

The increasing complexity of transaction processing systems, such as databases, led to the development of self-adaptive [31, 33], self-tuning [11, 75], and self-managing systems [20, 41]. The eventual objective of this discipline is to create a self-driving system that can autonomously predict workload and network changes, and reconfigure itself to optimal performance without human intervention. Research is currently underway to achieve this goal [39, 40, 45–47, 57, 58, 76]. Since blockchains have evolved to support complex transactions using smart contracts, they are now categorized as transaction processing systems [63]. This raises the question of whether self-driving blockchains are feasible.

Blockchain systems, like other transaction processing systems, require dynamic reconfiguration to cope with changes in workloads and network conditions [9, 43, 69]. However, optimal reconfiguration is particularly challenging due to the complexity of the blockchain stack, which has diverse configurable parameters [10, 43]. These include administrative policies, database definitions, consensus protocols, ledger settings, and smart contract design, among others [9, 42, 62, 72]. Therefore, domain expertise is essential for achieving optimal performance in blockchain systems. However, this expertise comes at a high cost, with the estimated maintenance cost of a blockchain application being up to 25% of the total development cost [36, 67, 68]. As long as blockchains remain complex and expensive, enterprises will hesitate to adopt this platform for their use cases.

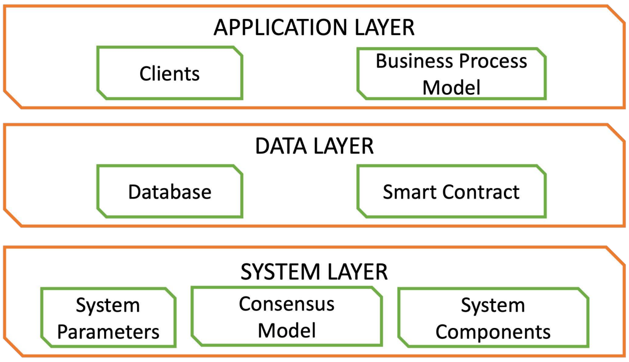


Figure 1: Adaptable Features

Consequently, a self-driving blockchain that can eliminate human involvement is desirable. Some initial steps in this direction have been taken through the development of self-adaptive and auto-tuning blockchain systems [43, 72, 73]. However, the existing systems are either not completely autonomous and hence require human intervention or are confined to tuning a single aspect of the blockchain stack. Unlike the database community, the blockchain literature has yet to explore the concept of a comprehensive self-driving blockchain extensively. Self-driving systems can either be created from scratch [58, 76] or by building upon existing systems [46]. The development of over 1000 different blockchain systems worldwide in a relatively short time span has made it challenging for enterprises to choose the ideal blockchain for their applications [26]. Rather than augmenting this problem, we focus on exploring self-driving possibilities in existing systems. Further, stakeholders of established systems may be reluctant to switch to a new blockchain platform and integrating autonomous capabilities into existing systems helps to avoid this requirement. Adapting the configuration settings of public blockchains may lead to hard forks as not all network participants may accept the changes [62]. In contrast, permissioned blockchains, which are more commonly used by enterprises, would welcome such changes if they can improve the overall performance. Hyperledger Fabric is one of the most popular permissioned blockchains with over 50 enterprise partners [25, 60].

This paper delves into the opportunities for self-drive in Hyperledger Fabric. Our investigation involves identifying *adaptable features*, i.e., the parameters and components of the blockchain stack that can be dynamically tuned to improve performance, as illustrated in Figure 1. We address specific challenges and offer a blockchain perspective on the topic of self-driving systems. Furthermore, we implement a prototype and conduct experiments to evaluate the feasibility of our findings.

Our contributions can be summarized as follows:

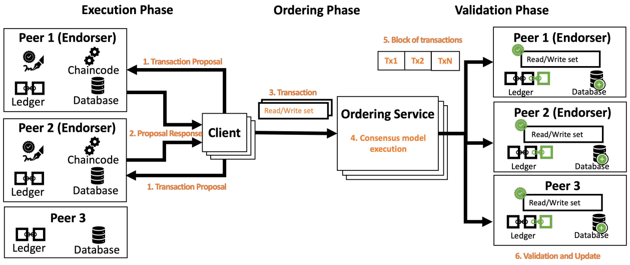


Figure 2: Transaction Flow in Hyperledger Fabric

(1) We scrutinize the various dynamic aspects of the blockchain environment, such as network and workload evolution, to determine the necessity of self-driving blockchains. We compare and contrast our findings with existing research on databases and highlight aspects unique to blockchains. Our goal is to encourage the blockchain research community to contribute more to the development of self-driving blockchains.

(2) We conduct a thorough analysis of the entire system stack of Hyperledger Fabric and identify specific parameters and components, which we call *adaptable features*, that are suitable for autonomous adaptation. Our study also highlights the challenges in making these adaptable features autonomous and provides potential solutions. By doing so, we encourage the blockchain community to explore self-driving opportunities within existing systems instead of creating a new blockchain from scratch for each new use case.

(3) We evaluate our findings by setting up three demonstrative autonomous systems, each targeting a different level in the blockchain stack. Our results indicate up to 11% improvement in success throughput and 30% decrease in latency. This is a significant first step towards implementing a fully autonomous system in the future. To the best of our knowledge, this is the first comprehensive discussion and evaluation of self-driving blockchains.

(4) The implementation of our systems, experimental workloads, and trained models are made available as open source. The research community can use these resources as a foundation to conduct further investigations with other adaptable features and machine learning strategies. This will facilitate further discussions on self-driving blockchains.

2 Hyperledger Fabric

One of the most popular open-source permissioned blockchain systems, Fabric [2], was established under the Linux Foundation. Fabric is unique in that it allows for smart contracts to be created using general-purpose languages. This gives clients the ability to submit transactions to a decentralized network, where control is shared among multiple entities instead of a single trusted entity. All potential functions that can be executed within a transaction are defined in a smart contract, referred to as chaincode. The system maintains a distributed, versioned key-value store known as the world state.

Each key in the network has a version number that gets updated with every write. The platform maintains a comprehensive history of all transactions, both successful and failed, through a distributed ledger that groups transactions into blocks. This distributed ledger, along with the world state, gets replicated on a set of distributed nodes called peers that are registered on the Fabric network. The peers receive blocks of transactions from an ordering service that guarantees ordered delivery and validate each transaction independently before updating their copy of the world state and the ledger.

Endorsers are a subset of peers that not only validate transactions but also endorse and execute the transactions. The number of endorsements required for a transaction to be accepted as valid is defined by an endorsement policy. Furthermore, peers are grouped into organizations, which typically correspond to real organizations or branches of an enterprise. These organizations play an important role in the endorsement policy as such policies can specify the number of endorsements required from each organization's respective peers. In Fabric, the transaction flow consists of three main phases: execution, ordering, and validation. This process is commonly known as the Execute-Order-Validate (E-O-V) model, and it is visualized in Figure 2.

Execution Phase. When clients need to update data stored in the blockchain, they initiate a transaction proposal to the endorsers. This transaction can include multiple requests for reads and writes to one or more keys in the world state. The endorsers simulate the transaction's execution on the current world state, generating a read/write list for each key involved in the transaction. They then send a response to the client, which includes their signature and the read/write set. During the transaction flow, the client collects responses from the endorsing peers and forwards them to the ordering service nodes.

Ordering Phase. The ordering service uses a consensus protocol, such as Raft [55] or BFT [5], to order transactions received from the clients. A transaction block is created based on three conditions: if a fixed duration of time has elapsed (block timeout), if a fixed number of transactions has been received (maximum message count), or if the total size of transactions has reached a fixed limit (maximum preferred bytes). The ordered block of transactions is then sent to all peers.

Validation Phase. When the ordering service sends a block of transactions, each peer validates every transaction in the block independently. Each peer verifies if a sufficient number of valid endorsing peer signatures, based on the endorsement policy, have been collected (Validation System Chaincode (VSCC) validation). After that, the peer checks if the version of each key in the read set of each transaction is equal to the version of the same key in the current world state (Multi-Version Concurrency Control (MVCC) validation). If both VSCC and MVCC validation checks pass, the write sets of the transactions are applied to the world state. However, if any of the validation checks fail, the client is informed that the transaction has been aborted, and the world state does not change. Once the validation is complete, the validated block containing both aborted and committed transactions is added to the ledger. The status of every transaction, whether committed or aborted, is logged for future reference.

3 Need for self-driving blockchains

In this section, we explore the reasons that necessitate the implementation of self-driving blockchains. Our analysis draws upon insights from the database literature while also shedding light on the unique concepts that are specific to blockchains.

3.1 Workload Evolution

Blockchains, similar to other transaction processing systems, have to handle frequently varying workloads. Depending on the application, workload variation may follow typical diurnal patterns, such as higher transactions during the day than night or spike patterns, such as sudden influx during Christmas in a supply chain management scenario [30, 57]. Apart from these familiar patterns, since blockchains are geographically distributed, the dynamic addition of participants from various time zones and their corresponding transactions can change the overall workload structure [51].

Unlike other transaction processing systems, blockchains also need to process administrative transactions apart from application-related transactions. For example, a *configuration transaction* needs to be executed when the system is reconfigured, such as changing the block size or integrating a new peer [2, 70]. Such transactions also follow the complete transaction lifecycle. Users may also trigger historical queries that read the complete or parts of the blockchain ledger to confirm the validity of their own transactions [66]. Such transactions are highly time-consuming.

When considering Fabric specifically, it has an optimistic concurrency control model where transactions can fail due to data dependency [64]. In such cases, the client may resend failed transactions immediately or later, depending on the business process logic of the enterprise. Additionally, Fabric's FIFO ordering strategy can result in situations where one type of transaction overwhelms the system, blocking all other transactions [30].

In summary, blockchains handle heterogeneous workloads that follow unpredictable arrival patterns and require the processing of additional administrative transactions, making them a complex system to manage. The optimal configuration of various system parameters, network components, smart contracts, database models, consensus algorithms, and business process models greatly depends on the workload [9, 10, 69]. Currently, only static auto-tuning systems are available for users to determine the best settings for their specific type of workload [43]. However, to use such systems, it is crucial to obtain appropriate representative workloads. This proves to be particularly challenging for permissioned blockchains, which are primarily used for enterprise purposes, since private organizations are hesitant to reveal their workloads. As a result, a self-driving blockchain that has the ability to monitor its evolving workload and dynamically adjusts itself to the ideal settings can be a promising solution.

3.2 Network Evolution

Scaling in blockchains is highly heterogeneous and dynamic [12, 65]. In permissioned blockchains, the different system components are mapped to physical entities in an enterprise. For example, the peer nodes of a Fabric network are grouped into administrative units called *organizations* that typically correspond to the physical organizations or branches of a company [2]. As a result, real-world administrative activities of enterprises, such as expanding their global reach or acquiring other organizations, necessitate adapting the blockchain network. Moreover, specific network components, such as endorsers and orderers, which have additional privileges, such as executing the smart contract and ordering the transactions, may be reassigned to different geographical locations depending on enterprise management changes in the physical world. System parameters must be adjusted to support such network scaling. For instance, if the number of ordering nodes is too high, communication costs increase, and performance is negatively affected. In such cases, Fabric recommends dynamically redesigning the network into subsets called channels and deploying separate ordering node sets per channel [16].

Additionally, even without any changes in the network components, the blockchain ledger grows perpetually over time. Therefore, the network must be constantly monitored and adjusted accordingly to avoid any bottlenecks. For example, Fabric recommends increasing the resources whenever the CPU, memory, or disk space usage reaches 70%, as high resource utilization significantly impacts performance [16]. However, since there are multiple distributed system components (peers, endorsers, orderers, clients, database, ledger), monitoring and identifying the bottleneck is challenging. Further, given the decentralized nature of blockchains, the participants need to reach consensus before taking scaling decisions. Consequently, a self-driving blockchain that constantly monitors the evolving network, identifies bottlenecks, triggers the consensus mechanism and automatically adapts the network configuration would be highly beneficial.

3.3 Performance and Fairness

Self-driving systems are designed to achieve multiple goals, including optimal performance in terms of throughput and latency. This is particularly true for self-driving transaction processing systems such as databases as well as blockchains [39, 40, 43, 45–47, 57, 58, 72, 73, 76]. The ability to sustain adequate throughput despite workload or network changes without (or with minimal) human intervention is the ultimate goal.

Another important objective for blockchains is fairness. Since blockchains lack centralized entities, transactions are generally processed in a first-in, first-out (FIFO) order, which may result in geographically closer and resource-intensive clients being able to commit more transactions [51]. In enterprise scenarios where many participants have equal administrative rights, such unequal representation of their transactions on the ledger may raise trust issues [7, 30]. To address this problem, a self-driving blockchain that identifies dominant clients, controls their transaction admission rates, and ensures fairness in the network is urgently needed. Such a system would ensure that all participants are equally represented on the ledger, thereby promoting trust and transparency.

4 Self-Driving Opportunities & Challenges

Since we have established the need for a self-driving blockchain in Section 3, our main objective now is to determine the feasibility of such a system. To achieve this, we must ascertain which components and configuration parameters require dynamic adaptation when the workload or network evolves. Some system configuration parameters that have an impact on the performance have been identified in the literature [43]. However, a majority of these parameters cannot be tuned without restarting the blockchain network, which is not possible in a live network. In contrast, we will examine specific features that can be adjusted without requiring a network restart and are thereby suitable for designing a self-driving blockchain. Additionally, we will solely concentrate on features that can be modified at runtime without significant alterations to the original system's architecture. We analyze the complete blockchain stack using experimentation and literature review to identify such adaptable features. In this section, we explain each of these features and discuss the challenges of dynamically adapting them. The experimental setup, workloads and metrics definition for Figures 3-5 can be found in Section 6.

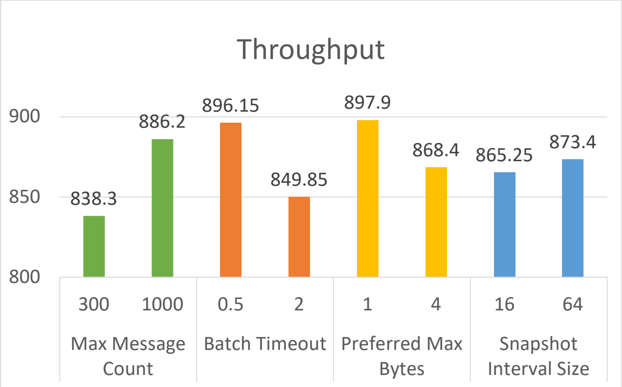


Figure 3: Impact of configuration parameters on performance

4.1 System Layer

We conducted an experimental analysis of various dynamically configurable system parameters to determine the adaptable features of Fabric at the system layer. The main results that highlight the impact of the parameters on overall throughput for a send rate of 1000 TPS are illustrated in Figure 3.

4.1.1 Max Message Count

The *max message count* refers to the maximum number of transactions allowed in a block. The ideal value for *max message count* varies based on the workload and the performance metric being considered. For instance, in our experiments, higher values of *max message count* significantly improve the overall throughput (cf. Figure 3). The literature recommends setting the *max message count* to match the incoming transaction rate of the workload [9, 16]. However, research has also shown that if the transaction rate is below a system's throughput saturation point, lower values for *max message count* are optimal [69]. Since all the experiments in the literature, as well as ours, are conducted with different workloads under different network conditions, we cannot derive a consistent relation between *max message count* and performance. Therefore, the *max message count* can be identified as an adaptable feature that needs to be dynamically adjusted based on the evolving workload and network.

4.1.2 Batch Timeout

The *batch timeout* is the maximum timeout after which a block is created with the currently available transactions. We observe that tuning this parameter has an impact on the performance. For example, when *batch timeout* is set to 2 seconds, which is the default value, it negatively impacts the overall throughput (cf. Figure 3). The official recommendation from Fabric is to set this value to *max message count* divided by the transaction rate [16]. Therefore, since *max message count* is an adaptable feature and the transaction rate is variable, *batch timeout* must be considered as an adaptable feature.

4.1.3 Preferred Max Bytes

The *preferred max bytes* refers to the maximum size (in MB) of all the transactions allowed in a block. Our findings indicate that the throughput is impacted by *preferred max bytes* (cf. Figure 3). Fabric recommends setting *preferred max bytes* to *max message count* multiplied by the average transaction size [16]. Since the optimal value of *preferred max bytes* depends on the incoming workload, it can be classified as an adaptable feature.

4.1.4 Snapshot Interval Size

The *snapshot interval size* parameter of the consensus protocol in Fabric (Raft) defines the number of bytes per which a snapshot of the log is taken. This is the only dynamically tunable parameter for Raft. While creating snapshots at regular intervals reduces disk space usage, it can be an expensive process [24]. Therefore, dynamically tuning this parameter based on the incoming load and disk usage can be helpful. We observe that an increase in *snapshot interval size* has a slight impact on the throughput (cf. Figure 3). Due to its dependency on the incoming workload, *snapshot interval size* can be identified as an adaptable feature

Lessons Learnt: Our experiments help to understand the impact of individual system configuration variables on the performance of Fabric. However, manually deriving the best combination of values for these parameters would be costly and brittle. This highlights the need for a self-driving blockchain. Self-driving systems generally adopt machine learning strategies, which involve exploring multiple values until the system learns the ideal setting. However, this process may face several obstacles, one of which is transaction queueing caused by the block size. In our experiments, we observed that creating multiple small blocks may overwhelm Fabric's ordering service when the transaction rate is high, causing the network to hang. Further, in our experiments, a very low *snapshot interval size* for a high transaction rate also led to system hang-ups. Hence, it is crucial to carefully choose the values of system parameters and transaction rates during the learning phase to prevent such issues in the Fabric network.

4.2 Data Layer

The data layer comprises the blockchain ledger, the smart contracts, and the database. In Fabric, the ledger and database cannot be dynamically reconfigured, so the focus is on optimizing the smart contract performance. Smart contracts play a pivotal role in the functioning of blockchains. To optimize their performance, various strategies are employed, such as delta writes, smart contract partitioning, and primary key redefinition [10]. However, the effectiveness of these strategies depends on the workload. For instance, *delta* writes can convert update transactions that increment a variable into write-only transactions, reducing transaction dependency failures in update workloads. However, this optimization strategy can negatively impact the performance of compute workloads. Therefore, adapting the smart contract according to the workload would be useful. To examine this, we created a smart contract that allows for a value to be incremented as an update transaction or as a write-only transaction using two different function implementations: *vanilla* and *delta*. We evaluate the performance impact of both implementations using an update workload and a compute workload. More details about the smart contract and the workloads can be found in Section 6.2.

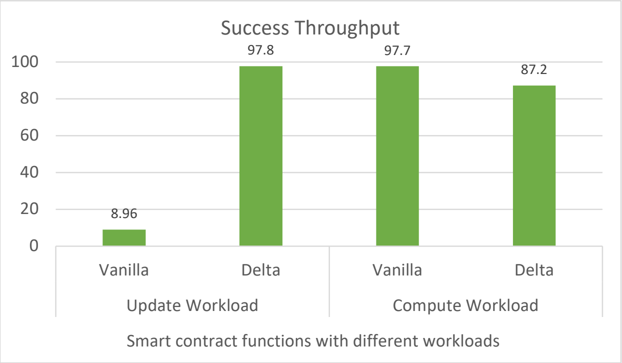


Figure 4: Impact of smart contract on performance

Our experimental results shown in Figure 4 indicate that the use of the *delta* implementation results in a significant improvement in success throughput for update workloads. However, for compute workloads the success throughput decreases with the use of the *delta* implementation. Further, over a longer duration, we observed that Fabric is unable to sustain the high latency resulting from the use of the *delta* implementation with compute workloads (not shown in Figure 4). In Fabric, smart contracts can be upgraded in real-time, making them an adaptable feature that can be customized to meet the specific requirements of different workloads.

Lessons Learnt: In enterprise scenarios, smart contracts are often defined as automated executions of contractual agreements between entities in the real world [23, 28]. As a result, frequent upgrades to smart contracts are generally discouraged. Our solution to this issue is to include multiple implementations of the logic within the same smart contract and selectively invoke the desired implementation from the client side based on the varying workload. This approach enables greater flexibility and reduces the need for frequent upgrades. Further, based on our experiments, we conclude that for a self-driving blockchain system, sufficient duration must be given for each learning step to correctly understand the effect of the adaptable features.

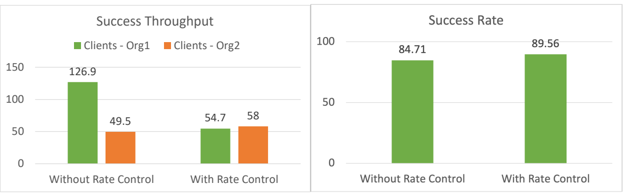


Figure 5: Impact of rate control on performance

4.3 Application Layer

The clients are the main components of the application layer, and fairness among the clients is crucial for building users' trust in a blockchain system. However, ensuring fairness is difficult due to the diversity in the geographical locations of the clients relative to endorsing peers and the resources available to the clients. There are instances where high transaction rates from a few clients congest the system, leaving the other clients with very low throughput [30]. Fabric's optimistic concurrency control strategy also follows a first-come-first-serve model, which can cause transactions from slower clients to fail more frequently. Monitoring such bottlenecks and dynamically adapting client transaction send rates is essential to ensure fairness. Therefore, the admission rate of clients is a potential adaptable feature.

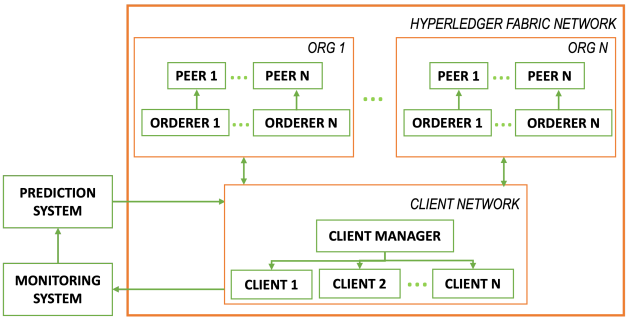


Figure 6: Self-driving Blockchain System

We conducted an experiment that simulates an unfair distribution of transactions, where clients of one organization (clients-Org1) have a higher transaction send rate than the clients of another organization (clients-Org2). Further, the transactions generated by clients-Org1 have key conflicts with the transactions generated by clients-Org2. More details about the workloads can be found in Section 6.3. From our experimental results shown in Figure 5, it is evident that rate control can ensure fairness, but it comes at the cost of degraded overall success throughput. However, we can see in Figure 5 that the total success rate improves. This experiment clearly demonstrates the tradeoff between fairness and overall performance.

Lessons Learnt: The main challenge in this direction is to design an optimal fairness strategy. If the transaction rate of faster clients is significantly restricted, it can have a severe impact on overall performance. Moreover, such restrictions may not even lead to a corresponding increase in the success rate of slower clients. Therefore, it is crucial to consider both fairness and overall performance when designing a self-driving system.

5 Self-driving Blockchain System Design

To demonstrate self-driving capabilities, we integrated a prediction system and a monitoring system with the Fabric network. The monitoring system extracts performance metrics from the client network: overall throughput (in TPS), success throughput (in TPS), average latency (in seconds), number of successful transactions per client and success rate per client. The prediction system controls various adaptable features of the Fabric network and is explained in detail in this section. Figure 6 provides a visualization of the architecture of our system.

Our prediction system uses a reinforcement learning-based approach to autonomously reconfigure the adaptable features. Reinforcement learning [38] is a machine learning strategy that involves the agent applying an action, observing the consequence of its action on the environment, receiving a reward based on the consequences, and altering its actions over time to maximize the reward. It is mainly used in scenarios where training data is not initially available, and it is popularly used in self-driving systems [43, 75, 76]. Permissioned blockchains are mainly used by enterprises that are generally hesitant to share their workloads and ledger contents due to privacy concerns. Therefore, there is a lack of publicly available training data, which makes reinforcement learning a suitable approach for our use case. We implement our prediction system using a reinforcement learning library called Deep Q Network (DQN) provided by Stablebaselines [21].

For a reinforcement learning agent (RL agent), three parameters need to be defined. The *state* is the environment that needs to be observed by the agent. The *action space* is a discrete or continuous set of actions that the agent is allowed to take. The *reward function* is a quantification of the reward that an agent receives based on the effect of its action on the state. The reward function depends on the performance expectations of the blockchain system user, which is often defined in a service level agreement. The definition for each of these parameters varies based on our target adaptable feature and is explained in the upcoming sections.

During each step of the learning process, the prediction system communicates with the Fabric network through its clients. It notifies the clients of any required changes in configuration, smart contract logic, or admission rate. To ensure that these changes are implemented and have an impact on the network, the learning process is paused for a while before moving on to the next step. The monitoring system provides performance information, which the prediction system uses to update its *state* and calculate its *reward.*

As a tangible initial step towards realizing a self-driving block chain, we demonstrate the effect of dynamically modifying the identified adaptable features. At the system level, we demonstrate parameter tuning, where the *max message count*, *batch timeout*, *preferred max bytes* and *snapshot interval size* are dynamically adapted. At the data level, we demonstrate the self-drive capability by dynamically adapting the smart contract implementation. Finally, client admission rate is tuned dynamically to understand the potential for self-driving at the application level. For our experiments, we designed three locally autonomous systems based on reinforcement learning, which can pave the way to a completely self-driving system in the future.

5.1 RL Agent for Parameter Tuning

This section describes our autonomous parameter tuning mechanism. The RL agent of the prediction system needs to dynamically learn the optimal values of four configuration parameters – *max message count* (M_C), *preferred max bytes* (P_B), *batch timeout* (B_T), and *snapshot interval size* (S_I), Therefore, the action space (A_S) of the RL agent consists of the set of values that each of these parameters can adopt. We chose these values based on intuition and previous research. The default value of *max message count* is 500. Studies show that it is optimal if the *max message count* matches the incoming transaction rate [9, 16], which in our experiments is 300, 500, and 1000 TPS. The default value of *preferred max bytes* and *snapshot interval size* is 2 MB and 16 MB, and in our experiments, values less than the default led to a significant decrease in the performance (even such that the system hangs). Therefore, we chose the default value as well as two values higher than the default. The default value of *batch timeout* is 2s, and increasing this value significantly increases the latency if none of the other parameters related to block size are fulfilled by the incoming transactions. Therefore, we chose the default value as well as two values lower than the default. The action space (A_S) for the RL agent is then the cross-product of all possible values for all four parameters.

$$\begin{aligned} M_C &= [300, 500, 1000] \\ P_B &= [2, 4, 16] \\ B_T &= [0.5, 1, 2] \\ S_I &= [16, 32, 64] \\ A_S &= [M_C \times P_B \times B_T \times S_I] \end{aligned}$$

The reward function (R_W) of an RL agent depends on the performance expectations of the blockchain system user. In this experimental setting, we assume that maximizing the throughput (T) of the Fabric network is the primary goal. The average transaction send rate (SR) of the clients vary with time. Therefore, we need to consider the throughput relative to the send rate.

$$R_W = \frac{T}{SR}$$

Similarly, since the impact of the agent's action is measured by the change in the throughput relative to the send rate, the *state* (S_T) or environment that the RL agent needs to observe is determined by these values.

$$S_T = [T, SR]$$

At every learning step, the RL agent picks an action (A) from the action space randomly or based on previous experience.

$$\begin{aligned} &A = [m_c, p_b, b_t, s_i] \\ &where\, m_c \in M_C, p_b \in P_B,\ b_t \in B_T,\ s_i \in S_I \end{aligned}$$

The chosen action is packaged as a *configuration transaction* and sent to the Fabric network via its clients. After this transaction is endorsed, ordered and validated by the peers in the network, the action is applied, i.e., the configuration parameters are updated, and the RL agent moves to the next learning step.

5.2 RL Agent for Smart Contract Adaptation

In this section, we discuss the mechanism for autonomous smart contract adaptation. The RL agent needs to dynamically learn the optimal smart contract implementation – *vanilla* and *delta*, which we represent as 0 and 1 in the action space(A_S) for the RL agent.

$$A_S = [0, 1]$$

Adapting the smart contract aims to improve the success throughput. As a result, the reward function (R_W) of the RL agent is defined by the success throughput (S_{UT}) relative to the send rate (SR).

$$R_W = \frac{S_{UT}}{SR}$$

Similarly, since the impact of the agent's action is measured by the change in the success throughput relative to the send rate, the *state* (S_T) is determined by these values.

$$S_T = [S_{UT}, SR]$$

The RL agent picks an action (A) which is either the *vanilla-update* or the *delta-update* implementation at every learning step.

$$A = [a_s]\, where\, a_s \in A_S$$

We created a new configuration file for the Fabric client to define the smart contract implementation. The client's invocation of the smart contract is dependent on this configuration file, which is updated by the RL agent with the chosen action.

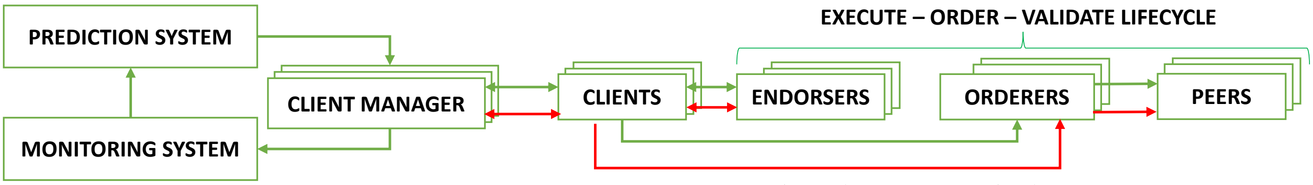


Figure 7: Life cycle of transactions generated in a Fabric network with (green) and without (red) the prediction system

5.3 RL Agent for Admission Rate Tuning

In this section, we delve into the mechanism of dynamic admission rate tuning with the aim of guaranteeing fairness across all clients. The RL agent needs to dynamically control the send rate of *clients-org1* ($S_R Org_1$) and *clients-org2* ($S_R Org_2$). The send rate cannot be increased, since in a real-world scenario, the clients would only send the required transactions and not create new transactions just to match a given send rate. Therefore, the send rate can either be throttled or kept unchanged. We define a decrease of 40% and 60% of the send rate based on intuition and experimentation. A decrease of less than 40% may not significantly influence the success rate of the other clients, and a decrease of more than 60% could hurt the overall throughput. Therefore, the send rates of *clients-org1* and *clients-org2* can either remain unchanged, decrease by 40% or decrease by 60%.

$$S_R Org_1 = [unchanged, decrease_{40}, decrease_{60}]$$
$$S_R Org_2 = [unchanged, decrease_{40}, decrease_{60}]$$
$$A_S = [S_R Org_1 \times S_R Org_2]$$

The main goal of this experiment is to ensure that the number of successful transactions generated by *clients-org1* ($S_{uc}T_r Org_1$) and the number of successful transactions generated by *clients-org2* ($S_{uc}T_r Org_2$) are as close to equal as possible. We use the Jain's fairness index ($\mathcal{J}_{fi}$) to quantitatively define fairness in the range (0,1], where higher values indicate a fairer distribution [37]. This index ($\mathcal{J}_{fi}$) is used to define the reward function.

$$\mathcal{J}_{fi} = \frac{(S_{uc}T_r Org_1 + S_{uc}T_r Org_2)^2}{2 * (S_{uc}T_r Org_1{}^2 + S_{uc}T_r Org_2{}^2)}$$

$$R_W = \mathcal{J}_{fi}$$

The impact of the agent's action is also measured by the change in the success throughput relative to the send rate and the fairness measure. Therefore, the *state* (S_T) is determined by these values.

$$S_T = [S_{UT}, SR, \mathcal{J}_{fi}]$$

The RL agent chooses an action (A) at every learning step which either decreases or maintains the admission rate of the clients.

$$A = [sr_1, sr_2]\, where\ sr_1 \in S_R Org_1,\ sr_2 \in S_R Org_2$$

We have developed a new configuration file for the Fabric client, which defines the transaction rate per organization. This configuration file is used to adapt the transaction rates of each organization. This file is updated by the RL agent with the chosen action at every learning step.

5.4 Decentralization Aspects

Our prediction system is designed as a centralized system that is independent from the blockchain network. This allows for easier implementation of updates, monitoring of the training process, and adherence to regulations and standards. Further, training systems typically result in high resource utilization over time. Since our prediction system is independent of the Fabric network, it can be deployed on a separate node with high resource allocation without encroaching on the resources required by the blockchain system. Additionally, such an independent prediction system can be readily replaced with different learning algorithms based on user requirements.

However, blockchains rely heavily on decentralized trust, which is an essential feature of their operation. Therefore, a self-driving blockchain also needs to maintain this decentralized nature. Though our prediction system is centralized the inherent decentralization properties of Fabric is maintained. The changes proposed by the prediction system are sent to a client manager, which then generates a *configuration transaction* for parameter tuning or a transaction with specific parameters to adapt the smart contract logic. The client manager also defines the transaction admission rate of each client. The client manager is a simulation of an automated business process execution system often used in enterprise scenarios to manage the execution of an application [71]. In a production-level setup, there could even be more than one client manager, and the prediction system would communicate the proposed changes to all of them. However, all transactions generated by the client manager undergo the complete execute-order-validate lifecycle defined by the Fabric network, i.e., all network participants need to reach a consensus when any change proposed by the prediction system is applied. As visualized in Figure 7, the lifecycle of transactions with and without the prediction system remains the same. Therefore, the decentralization and security guarantees of Fabric are maintained despite having a centralized prediction system.

6 Experimental Setup

We conducted our experiments on four clusters each with 1 master node and 3 worker nodes. All nodes are deployed with 100 GB memory and 30 GB storage. The master nodes have 32 virtual CPUs while the worker nodes have 16 virtual CPUs each. On all clusters, we launched a Fabric network on the workers with four peers (two per organization) and three orderers. Additionally, we deployed ten clients along with a client manager using Hyperledger Caliper, a benchmarking system for Hyperledger Fabric. We ran the prediction system and monitoring system on the master node of all clusters. We developed different workloads and smart contracts for evaluating the three autonomous systems as described in the following.

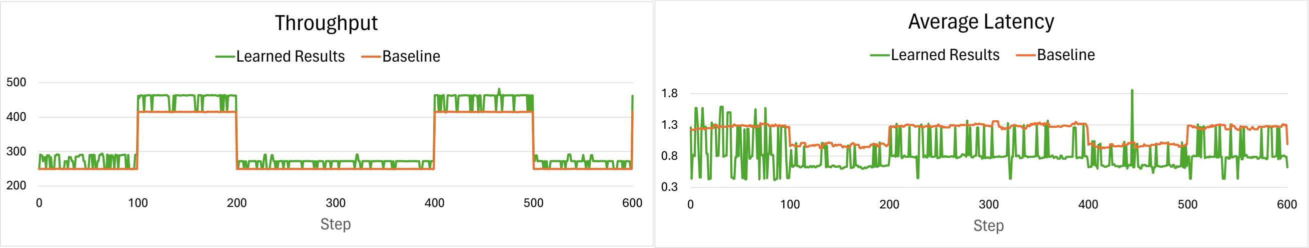


Figure 8: Update workload with and without autonomous parameter tuning

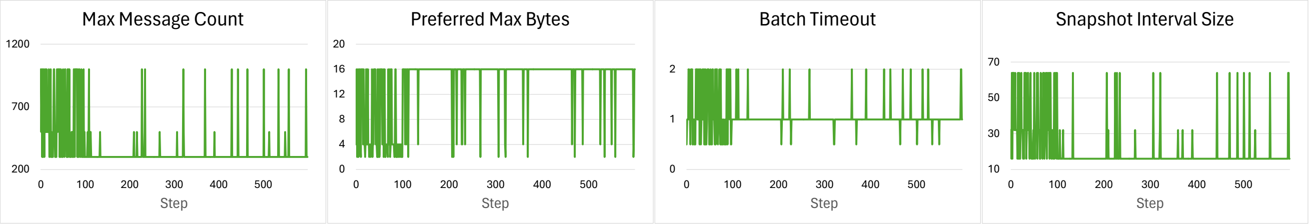


Figure 9: Values of parameters set by the prediction system for the update workload

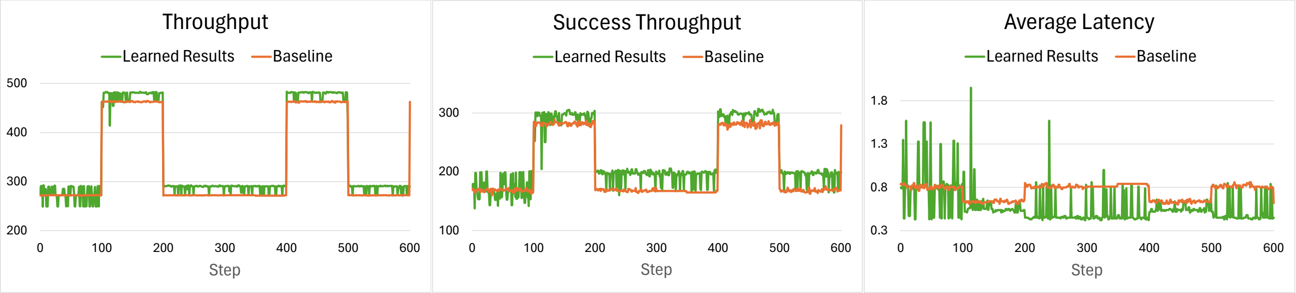


Figure 10: Skewed update workload with and without autonomous parameter tuning

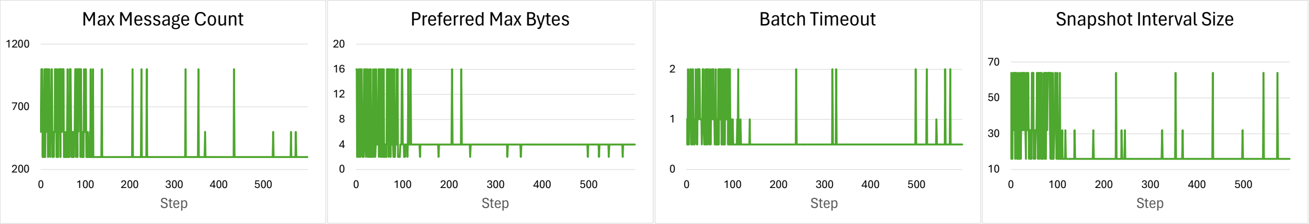


Figure 11: Values of parameters set by the prediction system for the skewed update workload

6.1 Workloads for Parameter Tuning

We use the *generator* smart contract from the literature [9], which can generate update transactions (single key read and write). Using this smart contract, we generate two different workloads: *update* workloads which have no transaction dependencies and *skewed update* workloads which have multiple transaction dependencies. We initialize the blockchain ledger with 10,000 keys. For the duration of our experiments, the clients' average transaction send rate oscillates between 300 TPS and 500 TPS at regular intervals of 100 steps to simulate an evolving workload. For all the experiments, the baseline is evaluated by setting the default value of 500, 2 MB, 2 seconds, and 16 MB for *max message count, preferred max bytes, batch timeout,* and *snapshot interval size,* respectively.

6.2 Workloads for Smart Contract Adaptation

We implemented a *music management* smart contract that includes the two main functions, *PlayMusic* and *CalculateRevenue*, from a music rights management scenario based on the literature [9]. In our database, every entry has a structure consisting of three components: *musicID*, *play-count*, and *total-revenue*. The *PlayMusic* function retrieves the entry for a given *musicID*, increments the value of the *play-count* by one, and writes it back to the database. This implementation of the function, which we call the *vanilla* implementation, results in an *update* transaction. The same function has an alternative implementation, the *delta* implementation, which writes a new entry into the database in the format *musicID-operation-value-transactionID* (for example, M01+1T01). In other words, every increment to the *play-count* creates a new entry in the database, making the transaction *write-only* instead of an *update*.

The *vanilla* implementation of *CalculateRevenue* function reads the value of *play-count* for a given *musicID* from the database. The *delta* implementation needs to aggregate the value of all keys in the database that have the partial format *musicid-operation-value-transactionID* (for example M01+1*) and add this aggregated value to the *play-count*. Such aggregations are time-consuming operations that, depending on the number of keys, can lead to system hang-ups for long-running experiments. Implementing additional smart contract functions that prune the delta keys and regularly invoking these functions as part of the workload can help resolve this issue. However, to reduce the complexity of the smart contract implementation and workload definition in our experimental setup, we simulate this aggregation functionality using a delay of 500ms. This simulation sufficiently demonstrates the effect of the *delta* implementation on performance while preventing disruptions to the experiment. Since our experiments focus on evaluating the prediction system's ability to learn the impact of adaptable features on performance, such a simulation is adequate. After aggregation, the *play-count* is multiplied by a constant and used to update the *total-revenue*.

We initialize the blockchain ledger with 10,000 keys. We generate an *update* workload that only invokes the *PlayMusic* function and a *compute* workload that only invokes the *CalculateRevenue* function. This simulates a real-world scenario where music is frequently played by all system users while the revenue is only occasionally calculated by the artists. The workload oscillates between *update* and *compute* to simulate an evolving workload. The average transaction send rate is 100 TPS. We also run two baselines. Baseline 1 uses only the *delta* implementation, while baseline 2 only uses the *vanilla* implementation.

6.3 Workloads for Admission Rate Tuning

We use the same *generator* smart contract and *skewed update* workload used for parameter tuning. We extended the client workload generation logic so that the transaction rates can be adapted per organization. Five of the clients registered to Org1 in the Fabric network send transactions at a rate of 250 TPS, and we call them *clients-org1*. The other five clients, which we call *clients-org2*, send transactions at a rate of 100 TPS. Further, the transactions generated by *clients-org1* have key conflicts with the transactions generated by *clients-org2*. This simulates a real-world scenario mentioned in the literature where one type of transaction floods the network, thereby causing the other transactions to fail [30].

7 Results and Observations

We conducted three sets of experiments to demonstrate our three locally autonomous systems. The results and observations from our experiments are described in this section.

7.1 Self-driven Parameter Tuning

This section describes our autonomous parameter tuning experiments. Figure 8 shows the overall throughput (which is equal to the success throughput since there are no failures) and average latency of the Fabric network with (learned results) and without (baseline) the prediction system on the *update* workload. The workload's send rate changes at every 100 steps, corresponding to the throughput changes visible in the graph. For the baseline, the values of the four configuration parameters are set to default. The results show that the performance of the network is improved significantly when the prediction system is employed. Specifically, the throughput increases by an average of 7%, and latency decreases by an average of 30%. This validates the positive impact of autonomous parameter tuning. From Figure 9 we observe that the most frequently used values that the prediction system learned over time for the *update* workload are (300, 16 MB, 1 s, 16 MB) for *max message count*, *preferred max bytes*, *batch timeout*, and *snapshot interval size*, respectively.

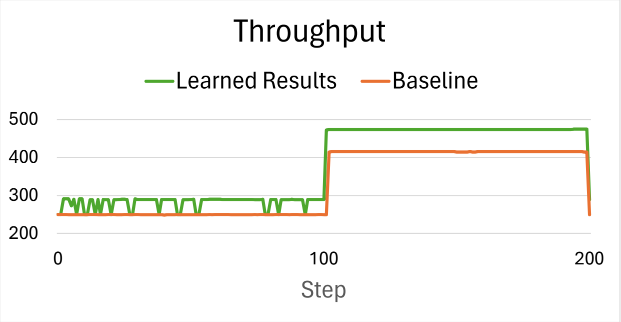


Figure 12: Autonomous parameter tuning with dynamically decreasing exploration rate

The experimental results with the *skewed update* workload are visualized in Figure 10. For the baseline, we used the best values (300, 16 MB, 1 s, 16 MB) for the configuration parameters from the previous experiment. We observe that even with the earlier learnt best values, the performance improves with the use of the prediction system. We observe an average of 4% increase in overall throughput, 8% increase in success throughput and 23% decrease in latency. From Figure 11 we observe that the most frequently used values that the prediction system learned over time for the *skewed update* workload are (300, 4 MB, 0.5 s, 16 MB) for *max message count*, *preferred max bytes*, *batch timeout*, and *snapshot interval size*, respectively. We observe that lower values are chosen for *preferred max bytes* and *batch timeout* with the *skewed update* workload. Comparing the baseline latency in Figure 8 and 10 we observe that transactions are processed faster with the *skewed update* workload. This could be due to the different configuration settings or because the presence of transaction failures reduce the number of writes on the database. Since the network is able to process transactions faster, the prediction system chooses to also create blocks faster by reducing the *preferred max bytes* and *batch timeout*.

It is evident that the two workloads have distinct optimal settings and despite our efforts to comprehend the rationale behind these specific selections, manually determining the perfect combination of values would be a daunting task, underscoring the significance of autonomous parameter tuning.

We further conducted an experiment with a dynamically decreasing exploration rate. Exploration rate defines the rate at which the prediction system tries new actions over time to identify the ideal setting. By decreasing this hyperparameter we observe in Figure 12 that a stable throughput that is on an average 11% higher than the baseline can be maintained. However, exploration is generally

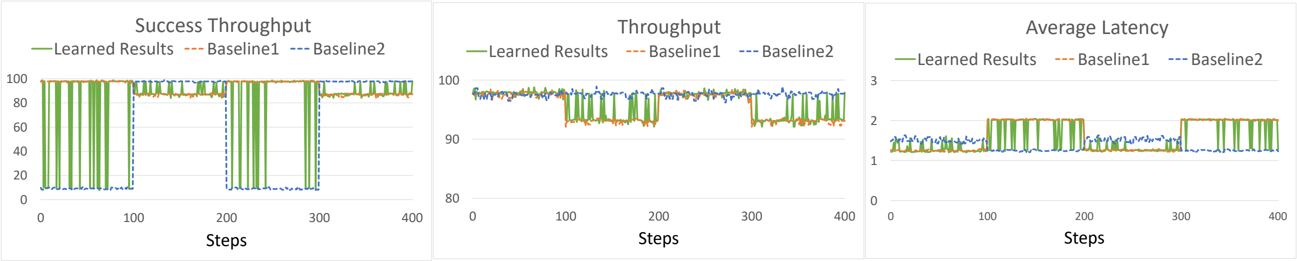


Figure 13: Performance with and without autonomous smart contract adaptation

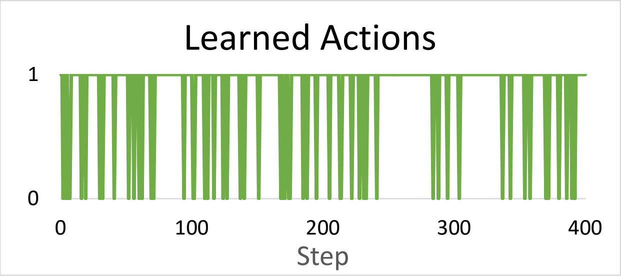


Figure 14: Learned smart contract tuning actions

encouraged in self-driving systems to accommodate unexpected changes that might require further learning.

7.2 Self-driven Smart Contract Adaptation

In this section, we discuss the experiments we conducted to enable automatic adaptation of the smart contract to explore the feasibility of autonomous smart contract upgrades. Figure 13 shows the overall throughput, success throughput, and average latency of the Fabric network with (learned results) and without (baseline) the prediction system using the *music management* smart contract. Baseline 1 uses only the *delta* implementation, while baseline 2 only uses the *vanilla* implementation. We initially use the *update* workload and then the workload oscillates between *update* and *compute* workloads at every 100 steps. We observe that Baseline 1 suffers from a very low success throughput of 9 TPS on average with the *update* workload but has high success throughput with the *compute* workload. In contrast, Baseline 2 performs better with the *update* workload and has a decrease in throughput with the *compute* workload.

The prediction system learns the performance impact of the two implementations on both workloads. Choosing the *delta* implementation over *vanilla* gives more than 10 times the success throughput with the *update* workload. At the same time, this choice decreases the success throughput only by an average of 10% with the *compute* workload. Figure 14 shows the actions learned by the prediction system where 0 represents the *vanilla* implementation and 1 represents *delta*. We observe that the prediction system correctly identifies that the *delta* smart contract implementation is a better choice for such an oscillating workload as the gain in performance with the *update* workload far exceeds the loss in performance with the *compute* workload. It chooses the *delta* smart contract implementation around 86% of the time over the *vanilla* implementation.

7.3 Self-driven Admission Rate Tuning

In this section, we delve into the experiments we carried out to adjust the rate at which the clients send transactions independently, with the aim of guaranteeing fairness across all clients. Figure 15 shows the Jain's fairness index, success throughput and success rate at each learning step with and without the prediction system.

In Figure 16, the values in the y-axis $[0, 0.5, 1]$ represent the actions $[\mathit{decrease}_{60}, \mathit{unchanged}, \mathit{decrease}_{40}]$ respectively. We observe that over time, the faster clients (*clients-org1*) learn to decrease the transaction send rate, while the slower clients (*clients-org2*) learn to maintain the default transaction send rate. The prediction system learns that decreasing the admission rate of the faster clients can result in a fair distribution of the number of successful transactions between *clients-org1* and *clients-org2* as shown by upto 16% increase in the Jain's fairness index in Figure 15.

We also observe that with the prediction system the slower clients have more successful transactions per learning step than the faster clients (not shown in the figure). However, this tradeoff is acceptable as the overall success rate improves by 6% (Figure 15). In other words, with the prediction system, the faster clients have a lower transaction send rate than the baseline but have a similar success rate to the baseline. The slower clients have a similar transaction send rate and success rate to the baseline, but the number of successful transactions per learning step increases (up to 16% increase, not shown in the figure). Further, there is an average of 30% decrease in overall success throughput (Figure 15), which is expected because the main objective is to ensure fairness between the clients in terms of successful transactions.

7.4 Learning Overheads

The prediction system is deployed on a separate node, and therefore, the overheads related to computing and storing the training model do not affect the Fabric components such as peers and orderers. In the parameter tuning experiments, the only interaction with the Fabric network is the execution of a *configuration transaction*. However, the network processes over 15,000 transactions at every step, so a single additional transaction will not have significant overhead. The prediction system updates the client configuration files for the smart contract and admission rate control experiments. The overhead of this file write operation is also insignificant as it happens in a non-blocking manner on separate process threads. Further, the metrics measured in our experiments are inclusive of all overheads that the prediction system might induce.

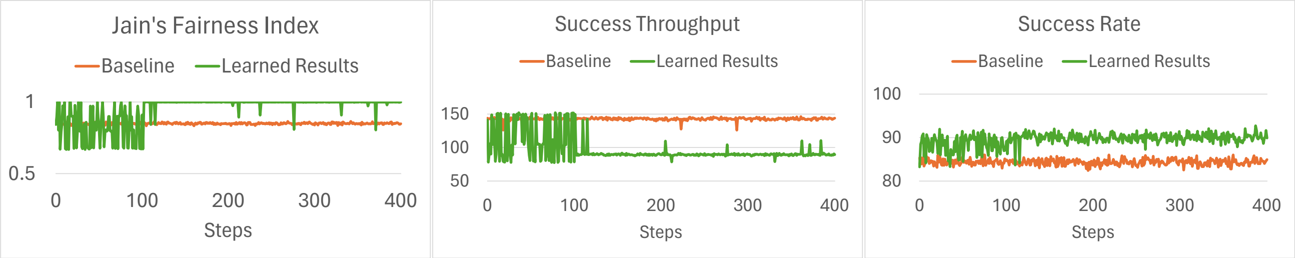


Figure 15: Performance with and without autonomous client admission rate tuning

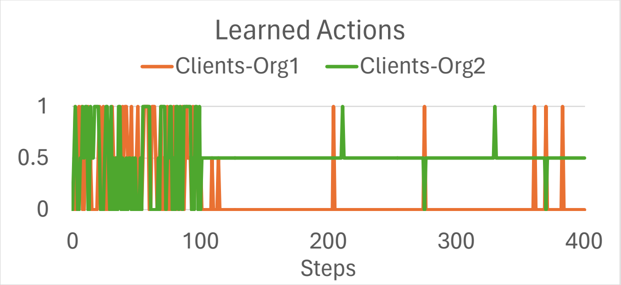


Figure 16: Learned admission rate tuning actions

7.5 Key Takeaways

We demonstrated that self-driving capabilities can be incorporated into different levels of the blockchain stack. Our experiments show that the prediction system is able to learn the optimal values for a given workload, leading to improved system performance. Additionally, reducing the exploration rate over time can help the prediction system to converge to these ideal values and produce stable, higher performance. However, this approach could hinder the detection of workload changes that require further learning. It is also important to note that fine-tuning the adaptable features can have varying impacts on each performance metric. For this reason, reward functions must be carefully formulated based on user requirements and considering the potential tradeoffs involved.

8 Discussions

We have identified several features that can be dynamically adapted and demonstrated their feasibility. Our findings represent a significant milestone in our quest to develop a completely self-driving blockchain. In this section, we will discuss further potential adaptable features, the transferability of our findings to other blockchain systems, and limitations of our work.

8.1 Further Adaptable Features

The adaptable features discussed in this paper so far are versatile and can be applied across various blockchain use cases without significant architectural changes. Additionally, our literature review has identified adaptable features that are more specific to particular use cases and require more extensive architectural modifications. We will discuss these features and their challenges in this section.

8.1.1 BFT Consensus Model

In enterprise use cases, blockchain participants are typically assumed to be non-byzantine, and the network only needs to be crash fault tolerant. However, recently, Fabric has introduced a Byzantine fault-tolerant consensus protocol [5]. The literature includes dynamic leader selection [74] and validator pool size adaptation [42] strategies for BFT protocols that could be included in a self-driving blockchain design. Further, the BFT consensus model in Fabric has 13 configuration parameters that can be tuned dynamically [14]. One such example is the *IncomingMessageBufferSize*, which is the size of the buffer that temporarily stores incoming transactions. Low values for this variable at high transaction rates cause bottlenecks [16]. Therefore, these configuration variables can also be identified as adaptable features.

8.1.2 Network Components

The performance of a Fabric network is greatly influenced by its network components. Research shows that the number and distribution of endorsers and orderers have a significant impact on network performance, and this can vary depending on the incoming workload [9, 10, 15, 16, 69]. Therefore, dynamically scaling these components up or down based on the workload can potentially improve network performance. However, since decentralized trust is a fundamental characteristic of blockchains, the network components are both geographically and administratively distributed among the participating entities. Geographically closer endorsers and orderers ensure faster transaction processing, and therefore, the decision on which components to adapt may not be unanimous. The process of updating these components dynamically involves transmitting a *configuration transaction* that all network members must endorse and verify according to a *system configuration endorsement policy.* To prevent the exclusion of any crucial decision-makers, a rigid endorsement policy must be established, and a manual override option must be implemented to reject any dynamic adaptation changes in the event of real-world conflicts among participants.

8.1.3 Database

Two pluggable database options are available in Fabric – LevelDB and CouchDB. LevelDB uses an LSM tree, which is suitable for write-heavy workloads, while B-Trees are better for read-heavy workloads [34]. By utilizing the concepts from the literature, a self-driving LevelDB that can switch between the two architectures would be perfect for an evolving workload [34]. Similarly, there are studies on automated schema design and parameter tuning for NoSQL databases, which can be used to develop a self-driving CouchDB [35, 48, 50]. Designing a self-driving database is a complex research problem on its own. The challenge multiplies when designing a self-driving blockchain alongside a self-driving database. Nonetheless, as the database community has already made significant progress in this field [39, 40, 45–47, 57, 58, 76], there is a possibility of having a self-driving database that could be seamlessly integrated with a self-driving blockchain.

8.1.4 Business Process Model

Permissioned blockchains are mainly used in enterprise settings where the execution of applications is typically based on a business process model. Studies indicate that the business process model is closely linked with the performance of the underlying system, and several optimization strategies can improve the system throughput [10, 27]. The activities in a business process model correspond to the transactions in a blockchain system. Therefore, the effectiveness of these optimization strategies is dependent on the workload. Research is moving towards self-adapting business process models [13, 56], making them suitable candidates for an *adaptable feature*. Redesigning a business process model often means redesigning the smart contract, which requires authorization from all decision-making entities in a blockchain network. As a result, frequent updates may not be recommended. To address this issue multiple designs could be included in the initial business model, and these can be selected dynamically by the workflow engine.

8.2 Technology Independence

It is challenging to have a generic discussion for a self-driving blockchain system due to the vast implementation differences between different blockchain systems. Therefore, we use Fabric as an example to support our discussions in this paper. Our work can serve as a foundation to hold similar discussions about other blockchain platforms. For instance, Corda and Multichain, which are two other popular permissioned blockchains, have more than 9 and 15 dynamically adaptable system parameters, respectively [18, 19, 52]. Corda utilizes notaries to endorse transactions, and distributing transactions across multiple notaries is expected to enhance throughput, similar to Fabric's endorsement policies [17]. Quorum's (another popular permissioned blockchain) mining frequency, or block time, affects transaction latencies in a linearly proportional manner, similar to Fabric's block size [4]. Additionally, optimization strategies for Solidity smart contracts, which are used by many different blockchain systems, are extensively discussed in the literature [1, 8, 53] and can be potential candidates for dynamic adaptation. Finally, fairness is a universal concept that can be applied to all blockchain systems [32, 44]. Therefore, the client transaction rate is an adaptable feature independent of the blockchain platform. Our paper highlights the need for a self-driving blockchain and demonstrates its feasibility on Fabric. Although it may not be viable to reuse the same model on other blockchains, the methodology we have presented can be applied to other blockchain platforms.

8.3 Limitations

In our experiments, we opted for the DQN learning strategy, which is a widely-used approach in self-driving systems [6, 54, 59, 61]. However, there are several other learning strategies that we could have considered to improve the performance of our approach. The literature mentions alternative approaches such as recurrent neural networks, linear regression, actor-critic model, and deep deterministic policy gradient for designing self-driving systems [43, 46, 57, 76]. Further, in our experiments, we use a single performance metric to define our reward functions to reduce the time and complexity of our learning approach. However, several optimized approaches to defining the reward function with multiple performance metrics can be found in the literature [22, 29, 49]. Nevertheless, our primary objective is to provide a blockchain perspective to the discussions on self-driving systems. As a result, we aim to identify adaptable features in blockchains and demonstrate their feasibility. Our focus is not to compare and determine the perfect learning approach for self-driving systems since this topic has already been extensively discussed in the literature [3, 43, 45].

9 Related Work

The database community has conducted extensive research on self-driving, self-managing, auto-tuning and self-adaptive database management systems that target several areas, such as resource allocation, configuration parameters, query optimization, partitioning, storage layout, and indexes [11, 20, 31, 33, 39–41, 45–47, 57, 58, 75, 76]. However, our conversation is centred around distributed ledger technologies that differ in architecture and use cases. For instance, the number and type of configuration parameters available for tuning, as well as the presence of smart contracts, set blockchains apart from databases. Additionally, our discussion also explores self-driving possibilities at the application level in terms of fairness, which is more significant for decentralized systems such as blockchains.

In the recent literature, there have been discussions on the subject of self-adaptive and auto-tuning blockchains. For instance, Adachain [72] is a self-adaptive blockchain that modifies its architecture according to the incoming workload to enhance performance. Contrastingly, we focus on applying self-driving strategies to existing systems without modifying their core architecture, which helps users of established blockchain systems to employ our methodology without the need to switch to a new blockchain platform. Athena [43] is an auto-tuning system that can tune the configuration parameters of a blockchain before deployment. Our focus, on the other hand, is on parameter tuning of a live network. Sabine [42] is a self-adaptive blockchain that adapts the number of validators in its consensus protocol, while Ursa [62] can adjust the number of transactions in a block based on user requirements. These works emphasize a single adaptable aspect of blockchains, whereas our paper focuses on identifying adaptable features throughout the entire blockchain stack. Further, self-driving possibilites at the application level have not yet been explored in the literature.

10 Conclusions

The demand for self-driving blockchain systems is growing due to the increasing complexity and cost of maintaining existing blockchain applications. While some initial steps have been taken towards creating self-adaptive and auto-tuning blockchain systems, a comprehensive self-driving blockchain has yet to be explored. This paper focused on the opportunities for self-drive in Hyperledger Fabric, one of the most popular permissioned blockchains used by enterprises. Our investigation identified adaptable features at different levels of the blockchain stack that can be dynamically tuned to improve performance. We also addressed specific challenges and possible solutions. We set up three demonstrative autonomous systems and conducted extensive experiments to evaluate the feasibility of our findings. The results suggest that self-driving blockchain systems are a promising avenue for future research.

References

[1] Elvira Albert, Jesús Correas, Pablo Gordillo, Guillermo Román-Díez, and Albert Rubio. 2020. Gasol: Gas analysis and optimization for ethereum smart contracts. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 118–125.

[2] 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. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. ACM, New York, NY, USA, Article 30, 15 pages. https://doi.org/10.1145/3190508.3190538

[3] Mrinal R. Bachute and Javed M. Subhedar. 2021. Autonomous Driving Architectures: Insights of Machine Learning and Deep Learning Algorithms. *Machine Learning with Applications* 6 (2021), 100164. https://doi.org/10.1016/j.mlwa.2021.100164

[4] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. 2018. Performance Evaluation of the Quorum Blockchain Platform. https://doi.org/10.48550/ARXIV.1809.03421

[5] Artem Barger, Yacov Manevich, Hagar Meir, and Yoav Tock. 2021. A Byzantine Fault-Tolerant Consensus Library for Hyperledger Fabric. arXiv:2107.06922 [cs.DC]

[6] Ayman Basheer, Hassan Jaleel Hassan, and Gaida Muttasher. 2021. Intelligent parameter tuning using deep Q-network for RED algorithm in adaptive queue management systems. In *International Conference on Micro-Electronics and Telecommunication Engineering*. Springer, 439–446.

[7] Nicolae Berendea, Hugues Mercier, Emanuel Onica, and Etienne Rivière. 2020. Fair and Efficient Gossip in Hyperledger Fabric. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. 190–200. https://doi.org/10.1109/ICDCS47774.2020.00027

[8] Tamara Brandstätter, Stefan Schulte, Jürgen Cito, and Michael Borkowski. 2020. Characterizing efficiency optimizations in solidity smart contracts. In *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 281–290.

[9] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. Association for Computing Machinery, New York, NY, USA, 221–234. https://doi.org/10.1145/3448016.3452823

[10] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2023. How To Optimize My Blockchain? A Multi-Level Recommendation Approach. *Proc. ACM Manag. Data* 1, 1, Article 24 (may 2023), 27 pages. https://doi.org/10.1145/3588704

[11] Surajit Chaudhuri and Vivek Narasayya. 2007. Self-Tuning Database Systems: A Decade of Progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria) *(VLDB '07)*. VLDB Endowment, 3–14.

[12] Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. 2018. Blockchain and Scalability. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 122–128. https://doi.org/10.1109/QRS-C.2018.00034

[13] Sihem Cherif, Raoudha Ben Djemaa, and Ikram Amous. 2015. SABPEL: Creating self-adaptive business processes. In *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*. 619–626. https://doi.org/10.1109/ICIS.2015.7166667

[14] Configuring and operating a BFT ordering service. 2023. https://hyperledger-fabric.readthedocs.io/en/latest/bft_configuration.html#channel-configuration

[15] Cluster considerations. 2023. https://hyperledger-fabric.readthedocs.io/en/latest/deployorderer/ordererplan.html#cluster-considerations

[16] Performance considerations. 2023. https://hyperledger-fabric.readthedocs.io/en/latest/performance.html

[17] Corda 2022. https://docs.r3.com/en/platform/corda/4.10/enterprise/key-concepts-notaries.html. [Online; accessed 12-April-2023].

[18] Corda: Network Parameters 2023. https://docs.r3.com/en/platform/corda/1.5/cenm/config-network-parameters.html. [Online; accessed 02-December-2023].

[19] Corda: Updating the network parameters 2023. https://docs.r3.com/en/platform/corda/1.5/cenm/updating-network-parameters.html. [Online; accessed 02-December-2023].

[20] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. 2013. ElasTraS: An Elastic, Scalable, and Self-Managing Transactional Database for the Cloud. 38, 1, Article 5 (apr 2013), 45 pages. https://doi.org/10.1145/2445583.2445588

[21] Stable Baselines Documentation. 2023. https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html

[22] Jonas Eschmann. 2021. *Reward Function Design in Reinforcement Learning*. Springer International Publishing, Cham, 25–33.

[23] Agata Ferreira. 2021. Regulating smart contracts: Legal revolution or simply evolution? *Telecommunications Policy* 45, 2 (2021), 102081. https://doi.org/10.1016/j.telpol.2020.102081

[24] Cloud Native Computing Foundation. 2023. https://etcd.io/docs/v3.5/tuning/

[25] Hyperledger Foundation. 2023. https://www.hyperledger.org/members

[26] Michael William G. 2023. https://watcher.guru/news/how-many-blockchains-are-there

[27] Luciano García-Bañuelos, Alexander Ponomarev, Marlon Dumas, and Ingo Weber. 2017. Optimized Execution of Business Processes on Blockchain. In *Business Process Management*, Josep Carmona, Gregor Engels, and Akhil Kumar (Eds.). Springer International Publishing, Cham, 130–146.

[28] Jack Gilcrest and Arthur Carvalho. 2018. Smart Contracts: Legal Considerations. In *2018 IEEE International Conference on Big Data (Big Data)*. 3277–3281. https://doi.org/10.1109/BigData.2018.8622584

[29] Adam Gleave, Michael Dennis, Shane Legg, Stuart Russell, and Jan Leike. 2020. Quantifying differences in reward functions. *arXiv preprint arXiv:2006.13900* (2020).

[30] Seep Goel, Abhishek Singh, Rachit Garg, Mudit Verma, and Praveen Jayachandran. 2018. Resource Fairness and Prioritization of Transactions in Permissioned Blockchain Systems (Industry Track). In *Proceedings of the 19th International Middleware Conference Industry* (Rennes, France) *(Middleware '18)*. ACM, New York, NY, USA, 46–53. https://doi.org/10.1145/3284028.3284035

[31] Michael Hammer and Arvola Chan. 1976. Index Selection in a Self-Adaptive Data Base Management System. In *Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data* (Washington, D.C.) *(SIGMOD '76)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/509383.509385

[32] Yuming Huang, Jing Tang, Qianhao Cong, Andrew Lim, and Jianliang Xu. 2021. Do the Rich Get Richer? Fairness Analysis for Blockchain Incentives. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 790–803. https://doi.org/10.1145/3448016.3457285

[33] Scott E. Hudson and Roger King. 1989. Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System. 14, 3 (sep 1989), 291–321. https://doi.org/10.1145/68012.68013

[34] Stratos Idreos, Niv Dayan, Wilson Qin, Mali Akmanalp, Sophie Hilgard, Andrew Ross, James Lennon, Varun Jain, Harshita Gupta, David Li, et al. 2019. Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn.. In *CIDR*.

[35] Stratos Idreos and Tim Kraska. 2019. From Auto-Tuning One Size Fits All to Self-Designed and Learned Data-Intensive Systems. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 2054–2059. https://doi.org/10.1145/3299869.3314034

[36] Anurag Jain. 2023. https://oyelabs.com/blockchain-app-development-cost/

[37] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. 1984. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA* 21 (1984).

[38] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[39] Jan Kossmann. 2018. Self-Driving: From General Purpose to Specialized DBMSs.. In *PhD@ VLDB*.

[40] Jan Kossmann and Rainer Schlosser. 2020. Self-driving database systems: a conceptual approach. *Distributed and Parallel Databases* 38 (2020), 795–817.

[41] Sushil Kumar. 2003. Oracle database 10g: The self-managing database. *White Paper* (2003).

[42] Guilain Leduc, Sylvain Kubler, and Jean-Philippe Georges. 2022. Sabine: Self-Adaptive BlockchaIn coNsEnsus. In *2022 9th International Conference on Future Internet of Things and Cloud (FiCloud)*. 234–240. https://doi.org/10.1109/FiCloud57274.2022.00039

[43] Mingxuan Li, Yazhe Wang, Shuai Ma, Chao Liu, Dongdong Huo, Yu Wang, and Zhen Xu. 2023. Auto-Tuning with Reinforcement Learning for Permissioned Blockchain Systems. *Proc. VLDB Endow.* 16, 5 (jan 2023), 1000–1012. https://doi.org/10.14778/3579075.3579076

[44] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. 2018. Toward fairness of cryptocurrency payments. *IEEE Security & Privacy* 16, 3 (2018), 81–89.

[45] Lin Ma. 2021. *Self-Driving Database Management Systems: Forecasting, Modeling, and Planning*. Ph. D. Dissertation. Carnegie Mellon University.

[46] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-Based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 631–645. https://doi.org/10.1145/3183713.3196908

[47] Lin Ma, William Zhang, Jie Jiao, Wuwen Wang, Matthew Butrovich, Wan Shen Lim, Prashanth Menon, and Andrew Pavlo. 2021. MB2: Decomposed Behavior Modeling for Self-Driving Database Management Systems. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1248–1261. https://doi.org/10.1145/3448016.3457276

[48] Ashraf Mahgoub, Paul Wood, Sachandhan Ganesh, Subrata Mitra, Wolfgang Gerlach, Travis Harrison, Folker Meyer, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. 2017. Rafiki: A Middleware for Parameter Tuning of NoSQL Datastores for Dynamic Metagenomics Workloads. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (Las Vegas, Nevada) *(Middleware '17)*. Association for Computing Machinery, New York, NY, USA, 28–40. https://doi.org/10.1145/3135974.3135991

[49] Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. 2006. Reward Function and Initial Values: Better Choices for Accelerated Goal-Directed Reinforcement Learning. In *Artificial Neural Networks – ICANN 2006*, Stefanos D. Kollias, Andreas Stafylopatis, Włodzisław Duch, and Erkki Oja (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 840–849.

[50] Michael J. Mior. 2014. Automated Schema Design for NoSQL Databases. In *Proceedings of the 2014 SIGMOD PhD Symposium* (Snowbird, Utah, USA) *(SIGMOD'14 PhD Symposium)*. Association for Computing Machinery, New York, NY, USA, 41–45. https://doi.org/10.1145/2602622.2602624

[51] Shashank Motepalli and Hans-Arno Jacobsen. 2023. Analyzing Geospatial Distribution in Blockchains. arXiv:2305.17771 [cs.DC]

[52] MultiChain runtime parameters 2023. https://www.multichain.com/developers/runtime-parameters/. [Online; accessed 02-December-2023].

[53] Quang-Thang Nguyen, Bao Son Do, Thi Tam Nguyen, and Ba-Lam Do. 2022. GasSaver: A Tool for Solidity Smart Contract Optimization. In *Proceedings of the Fourth ACM International Symposium on Blockchain and Secure Critical Infrastructure*. 125–134.

[54] Takafumi Okuyama, Tad Gonsalves, and Jaychand Upadhay. 2018. Autonomous driving system based on deep q learnig. In *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*. IEEE, 201–205.

[55] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Philadelphia, PA) *(USENIX ATC'14)*. USENIX Association, Berkeley, CA, USA, 305–320. http://dl.acm.org/citation.cfm?id=2643634.2643666

[56] Jamila Oukharijane, I Ben Said, Mohamed Amine Chaâbane, Rafik Bouaziz, and Eric Andonoff. 2018. A survey of self-adaptive business processes. In *Int. Business Information Management Association Conference, Seville, Spain*. 1388–1403.

[57] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. 2017. Self-Driving Database Management Systems.. In *CIDR*, Vol. 4. 1.

[58] Andrew Pavlo, Matthew Butrovich, Lin Ma, Prashanth Menon, Wan Shen Lim, Dana Van Aken, and William Zhang. 2021. Make Your Database System Dream of Electric Sheep: Towards Self-Driving Operation. *Proc. VLDB Endow.* 14, 12 (jul 2021), 3211–3221. https://doi.org/10.14778/3476311.3476411

[59] Baiyu Peng, Qi Sun, Shengbo Eben Li, Dongsuk Kum, Yuming Yin, Junqing Wei, and Tianyu Gu. 2021. End-to-end autonomous driving through dueling double deep Q-network. *Automotive Innovation* 4 (2021), 328–337.

[60] Michel Rauchs, Apolline Blandin, Keith Bear, and Stephen B McKeon. 2019. 2nd global enterprise blockchain benchmarking study. *Available at SSRN 3461765* (2019).

[61] Max Peter Ronecker and Yuan Zhu. 2019. Deep Q-network based decision making for autonomous driving. In *2019 3rd international conference on robotics and automation sciences (ICRAS)*. IEEE, 154–160.

[62] Na Ruan, Dongli Zhou, and Weijia Jia. 2020. Ursa: Robust Performance for Nakamoto Consensus with Self-Adaptive Throughput. *ACM Trans. Internet Technol.* 20, 4, Article 41 (nov 2020), 26 pages. https://doi.org/10.1145/3412341

[63] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. 2021. *Blockchains vs. Distributed Databases: Dichotomy and Fusion.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3448016.3452789

[64] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A Transactional Perspective on Execute-Order-Validate Blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 543–557. https://doi.org/10.1145/3318464.3389693

[65] Abdurrashid Ibrahim Sanka and Ray C.C. Cheung. 2021. A systematic review of blockchain scalability: Issues, solutions, analysis and future research. *Journal of Network and Computer Applications* 195 (2021), 103232. https://doi.org/10.1016/j.jnca.2021.103232

[66] Application side Programming Model. 2023. https://hyperledger-fabric.readthedocs.io/it/latest/Fabric-FAQ.html

[67] Sudeep Srivastava. 2023. https://appinventiv.com/guide/blockchain-app-development-cost/

[68] Akash Takyar. 2023. https://www.leewayhertz.com/cost-of-blockchain-implementation

[69] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. 2018. 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)*. 264–276. https://doi.org/10.1109/MASCOTS.2018.00034

[70] Updating a channel configuration 2022. https://hyperledger-fabric.readthedocs.io/en/release-2.2/config_update.html. [Online; accessed 12-April-2023].

[71] Meir Wahnon. 2024. https://github.com/meirwah/awesome-workflow-engines

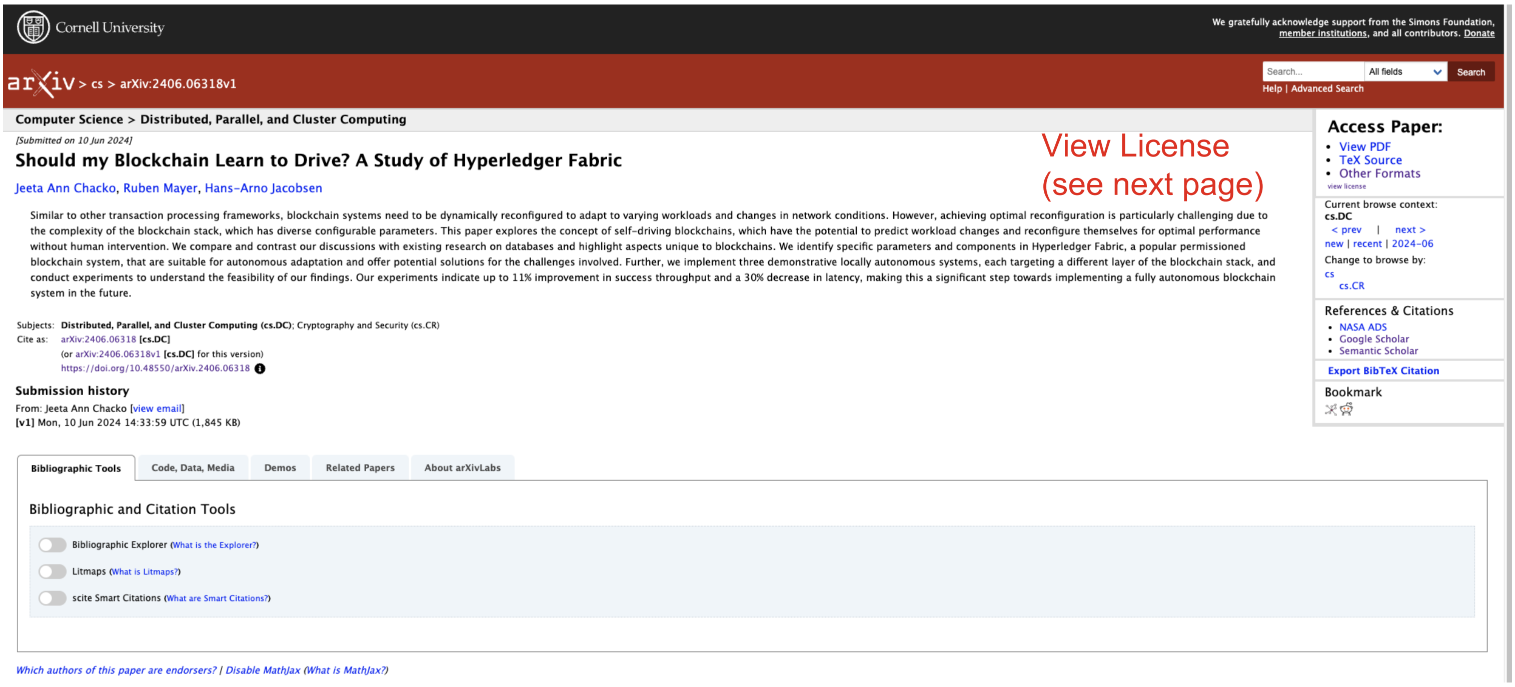
[72] Chenyuan Wu, Bhavana Mehta, Mohammad Javad Amiri, Ryan Marcus, and Boon Thau Loo. 2023. AdaChain: A Learned Adaptive Blockchain. *Proc. VLDB Endow.* 16, 8 (jun 2023), 2033–2046. https://doi.org/10.14778/3594512.3594531

[73] Jie Xu, Qingyuan Xie, Sen Peng, Cong Wang, and Xiaohua Jia. 2023. AdaptChain: Adaptive Scaling Blockchain With Transaction Deduplication. *IEEE Transactions on Parallel and Distributed Systems* 34, 6 (2023), 1909–1922. https://doi.org/10.1109/TPDS.2023.3267071

[74] Gengrui Zhang, Fei Pan, Sofia Tijanic, and Hans-Arno Jacobsen. 2023. PrestigeBFT: Revolutionizing View Changes in BFT Consensus Algorithms with Reputation Mechanisms. *arXiv preprint arXiv:2307.08154* (2023).

[75] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 415–432. https://doi.org/10.1145/3299869.3300085

[76] Xuanhe Zhou, Lianyuan Jin, Ji Sun, Xinyang Zhao, Xiang Yu, Jianhua Feng, Shifu Li, Tianqing Wang, Kun Li, and Luyang Liu. 2021. DBMind: A Self-Driving Platform in OpenGauss. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2743–2746. https://doi.org/10.14778/3476311.3476334



arXiv > cs > arXiv:2406.06318v1

Computer Science > Distributed, Parallel, and Cluster Computing

[Submitted on 10 Jun 2024]

View License
(see next page)

Should my Blockchain Learn to Drive? A Study of Hyperledger Fabric

Jeeta Ann Chacko, Ruben Mayer, Hans-Arno Jacobsen

Similar to other transaction processing frameworks, blockchain systems need to be dynamically reconfigured to adapt to varying workloads and changes in network conditions. However, achieving optimal reconfiguration is particularly challenging due to the complexity of the blockchain stack, which has diverse configurable parameters. This paper explores the concept of self-driving blockchains, which have the potential to predict workload changes and reconfigure themselves for optimal performance without human intervention. We compare and contrast our discussions with existing research on databases and highlight aspects unique to blockchains. We identify specific parameters and components in Hyperledger Fabric, a popular permissioned blockchain system, that are suitable for autonomous adaptation and offer potential solutions for the challenges involved. Further, we implement three demonstrative locally autonomous systems, each targeting a different layer of the blockchain stack, and conduct experiments to understand the feasibility of our findings. Our experiments indicate up to 11% improvement in success throughput and a 30% decrease in latency, making this a significant step towards implementing a fully autonomous blockchain system in the future.

Subjects:	**Distributed, Parallel, and Cluster Computing (cs.DC)**; Cryptography and Security (cs.CR)
Cite as:	arXiv:2406.06318 **[cs.DC]**
	(or arXiv:2406.06318v1 **[cs.DC]** for this version)
	https://doi.org/10.48550/arXiv.2406.06318

Submission history

From: Jeeta Ann Chacko [view email]
[v1] Mon, 10 Jun 2024 14:33:59 UTC (1,845 KB)

Bibliographic and Citation Tools

- Bibliographic Explorer (What is the Explorer?)
- Litmaps (What is Litmaps?)
- scite Smart Citations (What are Smart Citations?)

Which authors of this paper are endorsers? | Disable MathJax (What is MathJax?)

Access Paper:

- View PDF
- TeX Source
- Other Formats

view license

Current browse context:
cs.DC
< prev | next >
new | recent | 2024-06
Change to browse by:
cs
cs.CR

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

Export BibTeX Citation

Bookmark

arXiv.org - Non-exclusive license to distribute

The URI http://arxiv.org/licenses/nonexclusive-distrib/1.0/ is used to record the fact that the submitter granted the following license to arXiv.org on submission of an article:

- I grant arXiv.org a perpetual, non-exclusive license to distribute this article.
- I certify that I have the right to grant this license.
- I understand that submissions cannot be completely removed once accepted.
- I understand that arXiv.org reserves the right to reclassify or reject any submission.

Revision history

2004-01-16 - License above introduced as part of arXiv submission process
2007-06-21 - This HTML page created

Contact

Appendix D

A Comprehensive Study on Benchmarking Permissioned Blockchains

A Comprehensive Study on Benchmarking Permissioned Blockchains

Jeeta Ann Chacko[1], Ruben Mayer[2], Alan Fekete[3], Vincent Gramoli[3], and Hans-Arno Jacobsen[4]

[1] Technical University of Munich
chacko@in.tum.de
[2] University of Bayreuth
ruben.mayer@uni-bayreuth.de
[3] University of Sydney
{alan.fekete,vincent.gramoli}@sydney.edu.au
[4] University of Toronto
jacobsen@eecg.toronto.edu

Abstract. Blockchain benchmarking systems are actively discussed in the literature, focusing on increasing the number of blockchains that can be supported. However, the constant inception of new blockchains into the market and their vast implementation differences make it a massive engineering challenge. We provide a general discussion on the main aspects of benchmarking blockchains, highlighting the necessary contributions from the developers and users of blockchains and benchmarking systems. We identify problem statements across four benchmarking factors by investigating five popular permissioned blockchains. Further, we define a broad methodology to tackle these problems. We conduct a case study of five existing blockchain benchmarking systems for our evaluation and identify their limitations, clarifying the need for our methodology.

Keywords: permissioned blockchains · benchmarking systems

1 Introduction

Though blockchains were initially considered digital currency exchange systems, introducing smart contracts led to the classification of blockchains as decentralized transactional management systems that could support more use cases [64]. Later, the conception of permissioned blockchains that restricted the network access to authorized users and improved the overall performance made blockchains attractive for enterprise use cases. Currently, the most popular permissioned blockchain platforms, such as Fabric, Corda, Multichain, and Quorum, have around 30 to 70 enterprise partners using their systems for various use cases, such as banking, supply chain transparency, and digital asset management [20, 42, 54, 59].

However, the plethora of blockchain systems currently available in the market creates uncertainty in the selection process. A recent survey shows that 26% of users switched from their initially chosen blockchain at a later stage of development and that performance is one of the top selection criteria for blockchains [62]. Though most blockchains report their individual performance data, the vast differences in implementation, system configuration, and workloads make a fair comparison challenging [33]. This highlights the demand for a comprehensive and impartial blockchain benchmarking approach. Currently, there are multiple benchmarking system implementations available for blockchains [11,28,33,40,57, 65]. Each of them targets one or a specific set of blockchains to benchmark. The current focus in this research space is on increasing the number of blockchains supported by a benchmarking system. For example, Blockbench [28], the first benchmarking system for permissioned blockchains, supports four blockchains, while Diablo [33] and Gromit [57], the latest benchmarking systems, support seven blockchains. However, the rapid inception of new blockchains into the market makes this a massive engineering challenge.

Additionally, as is the case with most transaction processing systems, initially, the lines between the design and implementation of benchmarking systems are often blurred [4]. A well-implemented benchmarking system may still fail to consider all crucial aspects of benchmarking due to poor design [34]. For example, many existing benchmarking systems only support simple asset transfer scenarios [57, 65], while in reality, blockchains are employed for numerous other use cases. Therefore, we identify the need for a thorough discussion on the different aspects of benchmarking blockchains, which will assist in implementing a comprehensive and extensible benchmarking system in the future.

One needs to understand the similarities and differences between the various blockchain platforms to identify the diverse factors of benchmarking accurately. A significant challenge in this direction is the insufficient scientific literature. Since many blockchains are commercialized, apart from research papers, we must also analyze technical documentation and blog posts from the respective blockchain developers to understand their systems thoroughly. Further, given the vast implementation distinctions among the different blockchain systems, discussions regarding blockchain benchmarking should not be limited to developers of benchmarking systems, but should also include developers of blockchain systems.

Our discussions address the problems regarding crucial benchmarking elements such as system configuration, parameter tuning, workloads, and metrics. We emphasize the importance of these issues by extensively analyzing five different permissioned blockchains. We then define the contributions required from the entire blockchain community to tackle them. We also conduct a case study of five existing benchmarking systems to identify their limitations and highlight the need for contributions. For example, we identify various system configuration settings that affect the performance of each of the multiple blockchains, while current benchmarking studies only employ the default value for these settings. In detail, we provide the following contributions:

1. We formulate problem statements across four aspects of benchmarking based on five different permissioned blockchains (Fabric, Corda, Multichain, Quo-

rum and Diem). This highlights the importance of these problems across different blockchain platforms.
2. We define a general methodology to tackle the problems that spans across developers and users of blockchains as well as benchmarking systems. This highlights the contributions required from each of them to improve the domain of blockchain benchmarking.
3. We provide a case study of five different blockchain benchmarking systems and the corresponding benchmarking studies to highlight the current limitations. This can help benchmarking system developers to extend their implementations to adhere to our methodology.

2 Permissioned Blockchains

In permissioned blockchains, access is restricted to a set of authorized users, making them suitable for many enterprise use cases that cannot support anonymity. They are peer-to-peer networks with access controls operating on a distributed ledger. Despite being in the same classification, the multiple permissioned blockchain systems currently available have vast differences in their implementation. This section briefly overviews the basic concepts and transaction flow of five popular permissioned blockchains, accentuating their similarities and differences.

2.1 Hyperledger Fabric

Hyperledger Fabric (a.k.a Fabric) is an open-sourced, permissioned blockchain system under the Linux foundation [2]. Fabric follows an execute-order-validate (EOV) model, one of its unique features. The main components of a Fabric network are peers, endorsers, and the ordering service. Only the endorsers store the smart contracts, and transactions are sent to the endorsers for execution based on an endorsement policy. Speculative transaction execution results in a read-write set of all the keys in the transaction which is then forwarded to the ordering service. The ordering service is a cluster of nodes that employs the Raft consensus protocol to decide on the order of the transactions. Upon consensus, a block of ordered transactions is broadcasted to all peers. Every peer validates the speculative results of every transaction in the block with the current world state. After successful validation, the world state is updated, and the block of transactions is committed to the ledger.

2.2 Corda

R3 Corda is an open-sourced permissioned blockchain mainly designed for financial use cases [7]. In Corda, data is only shared among the network participants on a need-to-know basis. The nodes in a Corda network are authorized using an identity service. Further, a network map service is employed for node lookup, enabling point-to-point communication between nodes. An immutable object called a state describes any data known to the nodes at a specific point in time.

Each node has a vault or database that stores all the state sequences it knows. Constraints to ensure that a state is valid are defined using smart contracts. A transaction defines the input and output for a state transformation. Further legal prose can be attached to a transaction to settle future disputes, which makes Corda appealing to financial use cases. Notaries are specific nodes assigned with the responsibility of ensuring that output states are unique successors of input states thereby preventing double spending. When a transaction proposal is created, only the entities related to it execute the smart contract to ensure its validity. Further, notaries check each input state object in a transaction to ensure that they have not been consumed earlier and prevent double-spending. Transactions are committed after the transaction-related entities and the notaries sign them.

2.3 Multichain

Multichain is a fork of Bitcoin and shares many of its features [35]. However, it is designed for a permissioned environment where nodes prove their identity using a handshaking protocol when connecting to other nodes. Each node defines the public address for which it has a private key, and other nodes can send challenge messages to be signed with this key. Unlike Bitcoin, only a few nodes are granted mining privileges, and there is a single validator per block. The validator is scheduled in a round-robin style with tunable parameters. Other participants then execute the individual transactions in a block in the defined global order.

2.4 Quorum

Quorum is a fork of the Go implementation of Ethereum, where the P2P layer was redesigned to allow only authorized nodes [50]. A privacy layer is implemented to support private and public transactions in a permissioned environment. Quorum uses transaction managers to handle encrypted data, including an enclave, which is a hardware security module, to hold private keys. Private transactions are sent to transaction managers for encryption after verifying the sender, and only the entities related to the transaction can receive the decrypted data. Different protocols, such as Raft, IBFT, and QBFT, are employed to attain consensus in the Quorum network. When consensus is reached, all the nodes in the network execute the public transactions in a block, while private transactions are only executed by the entities related to the transaction.

2.5 Diem

Diem (earlier known as Libra) is a permissioned blockchain introduced by Facebook (now Meta) [27]. The network consists of two types of nodes - full nodes and validators. For every incoming transaction, validators check the signature, balance, and whether the transaction has been replayed, before sharing them with other validators. A BFT protocol (DiemBFT) is used to reach a consensus on the order of transactions. When a validator is elected as the leader, it proposes a block which is forwarded to the other validators for approval. Meanwhile, the transactions in the block are speculatively executed and also shared. Upon consensus, all the transactions of the proposed block are committed. Full nodes are employed to re-execute and store all transactions to provide evidence in the event of a history rewrite attempt. It ensures that validators cannot collude on transaction executions.

3 Benchmarking Guidelines

In this section, we focus on four important aspects to address when benchmarking permissioned blockchains. We consider examples from Fabric [2], Corda [37], Multichain [55], Quorum [60] (four of the most commonly used permissioned blockchains [62]) and Diem [27] when defining each problem statement. We then propose a general methodology for tackling each problem. The methodology targets blockchain developers, benchmarking system developers, as well as those conducting benchmarking studies on blockchains. We aim to bring to light the contributions required from each of them to the blockchain benchmarking space.

3.1 System Configuration

Problem Statement There is a vast distinction in the system components that compose the different permissioned blockchain implementations. The choice, count, and distribution of the different components significantly affect the performance. For example, the Hyperledger Fabric network consists of validating peers, endorsers, orderers, and clients where the count and distribution of each of these components impact the performance [9, 12, 38]. Corda offers two configurations for its notary nodes: validating and non-validating. Deploying multiple validating notary clusters can aid load balancing, improving performance [19]. In the Quorum network, performance is influenced by the choice between full nodes with a privacy manager or light nodes [32] for process-intensive tasks as well as boot nodes [14] or static nodes [15] for different peer discovery strategies. Multichain has the concept of data streams, and nodes that subscribe to these streams ensure faster information retrieval [52]. Also, Diem has the concept of validator nodes and full nodes, the choice of which can introduce additional overhead depending on the use case [26]. Therefore, identifying the influential system components and designing the optimal setup is crucial to ensure the best performance for each blockchain implementation. Further, even though the system configuration of each blockchain needs to be individually optimized, the hardware requirements or the hardware cost must be uniform across all blockchains for a fair benchmarking approach [61].

Methodology

1. Blockchain system developers need to provide extensive documentation and experimental results to quantify the influence of system components on the performance for each blockchain. Identifying and documenting a priority-based list of the main components that significantly impact the performance will be highly beneficial. Multichain published a list of tips for performance optimization on their website [53] which includes ideal server specifications, and though they do not provide concrete suggestions, this highlights the need for such documentation from the developers.
2. Benchmarking system users must design an optimal system setup specific to each blockchain based on their documentation. This is a challenging yet crucial task. Individually optimizing the system setup ensures benchmarking the best performing setup of each blockchain. Further, all the blockchains benchmarked together must employ uniform hardware or be limited to uniform hardware costs to ensure fairness [61].
3. Benchmarking system developers must support easy integration and reconfiguration of all system components. Due to the large number and type of components involved, system setup is often complex for blockchains [71]. Benchmarking systems need to provide automation scripts or at least detailed documentation that supports the integration of influential system components apart from the default to ease the system setup process. Further, the optimal system setup varies with use cases, so easy reconfiguration should also be supported.

3.2 Parameter Tuning

Problem Statement Parameter tuning is a significant factor to consider while benchmarking blockchains, and it is heavily discussed in the literature [9,47,70]. The literature mainly discusses generic parameters, such as block size, while system-specific parameters are largely ignored. However, both are equally important to ensure fair benchmarking. The number of transactions to include in a block is a well-known parameter that influences the performance of most blockchains [9,49], but Corda is an exception since the concept of blocks does not exist [18]. Further, there are system-specific parameters such as the set of cache-related parameters for GoQuorum [30,31] and Corda [17], the validator pool size, endorsement policy, and CouchDB parameters for Fabric [12], or the mining diversity and skip proof-of-work check [51] configurations in Multichain, all of which can be tuned for performance improvements. Also, Diem offers mempool [25] and consensus [24] configurations that are based on its unique implementation. Therefore, individually identifying and tuning the critical parameters for each blockchain is required to benchmark the ideal performance of every system. However, on the other hand, some parameters may impact the system's functionality and must be set equivalently to ensure a fair comparison. For example, Clique is byzantine fault tolerant with eventual finality, while Raft is only crash fault tolerant with immediate finality, and either can be chosen as the consensus protocol in Quorum [16]. If Fabric, which offers only the Raft consensus protocol, is benchmarked with Quorum, then to ensure fairness, Quorum's consensus protocol needs to be set to Raft. Parameter tuning is often discussed in the literature on benchmarking transaction processing systems [4, 34]. However, when considering blockchains, there is a more diversified set of parameters for tuning since the blockchain stack is comprised of numerous layers such as consensus models, access control protocols, database stores, smart contracts, and distributed ledgers.

Methodology

1. Blockchain developers should identify key parameters that influence the performance of their blockchain. They should ensure that all configuration parameters and quantitative evidence of their effect on performance are well documented. Workload-based analysis of these parameters should also be conducted and documented. Further, given the large number of parameters in blockchains, a prioritizing strategy would be beneficial. For example, Fabric has over 50 parameters, and a recent study quantitatively ranked the top parameters that significantly affect the performance [46].
2. Benchmarking system users must tune parameters based on the workload and system setup. Currently, benchmarking is often accomplished with the default parameter values or with a one-time tuning of limited parameters [28, 33,57]. However, studies show that parameter tuning significantly depends on the workload and system setup [9,47,70]. Therefore, parameter tuning should be done dependent on the use case that is being benchmarked. Recently, auto-tuning of blockchains is also being discussed in the literature, which could ease this process [46].

3.3 Workloads and Use Cases

Problem Statement The third important aspect to consider is the workload employed for benchmarking. Using existing workloads such as YCSB and TPC is a popular choice since these are well established in the community [28, 44]. However, blockchains often target different use cases than traditional transaction processing systems. Therefore, reusing existing workloads is often unrealistic and leads to inaccurate assumptions about the performance of a system [33]. But traditional workloads are still useful to benchmark scenarios where enterprises port their existing applications to blockchain systems. Further, blockchain implementations are varied, and each is designed with a specific use case in mind. For example, Fabric cannot handle highly skewed workloads due to its optimistic concurrency control model [9], and Corda supports only point-to-point requests between entities involved in a transaction [18]. Also, Quorum and Corda are mainly popular for financial use cases while Fabric applications range across multiple domains such as supply chain management and healthcare [73]. Further, the system setup, parameters, and transaction definition also vary with the use case. Multichain recommends different performance optimization strategies based on the expected type of workload [53], and Diem defines three different types of transactions based on the client account type [72].

Methodology

1. Benchmarking system developers should focus on both traditional as well as blockchain-specific workloads. Porting traditional workloads such as TPC and YCSB to blockchain environments is a good practice as it corresponds to scenarios where existing enterprise applications are migrated to blockchain platforms. However, the focus should also be given to blockchain-specific workloads, such as supply chain and digital asset management scenarios, to capture realistic performance capabilities better. Apart from workload generation, converting or porting the workloads to support multiple blockchain implementations is an important and challenging engineering task.
2. Benchmarking system developers must also generate system-specific workloads. Such workloads that stress test distinctive blockchains based on their specific design are essential to highlight accurate performance expectations. For example, private transactions in Fabric and point-to-point requests in Corda would need specific workloads different from other generic broadcast transactions. Also, the targeted use cases of each blockchain implementation should be supported.
3. Benchmarking system users and blockchain developers should provide use case-based discussion of benchmarking results. Benchmarking results will quantitatively indicate the most or least performant blockchain. However, a specific blockchain's intended use case must be considered before reaching a viable conclusion. For example, it has been quantitatively shown that Fabric is more performant than Diem [76]. However, Diem supports a byzantine fault-tolerant consensus protocol, while Fabric uses a crash fault-tolerant consensus protocol, both of which are suitable for entirely different use cases. Therefore, evaluation results need to be explored extensively in relation to the blockchain implementation and envisioned use.

3.4 Performance Metrics

Problem Statement The metrics used for benchmarking depend on the quality being benchmarked. Throughput and latency are the main client-visible metrics generally used in benchmarking blockchains when the focus is on performance; as well, some studies look at metrics reported from the blockchain platform, such as CPU usage or storage. However, there needs to be more clarity about how to define these metrics. Throughput is often defined as the number of transactions committed to a blockchain per second. However, for Fabric, failed transactions are also committed to the blockchain [2]. Latency is often described as the duration between transaction submission and final commit. However, submission time can be considered as the time the client submitted the transaction or the time the transaction entered the consensus protocol [28, 45]. Further, depending on whether the blockchain supports immediate or probabilistic finality, the definition of commit time changes [57]. Also, latency is a distribution, and single summary values such as mean or 95-percentile can be quoted, depending on what matters most for the specific use case. Therefore, a uniform definition for blockchain performance metrics is challenging. Further, system-specific metrics also need to be considered to provide a better understanding for the client. For example, apart from throughput and latency, Diem developers define a metric called *capacity* as "*the ability of the blockchain to store a large number of accounts*" [1].

Methodology

1. Blockchain developers must define generic as well as system-specific performance metrics. Generic metrics should either be uniformly defined for all blockchains along with the system-specific assumptions or be uniquely defined for each blockchain (or both). System-specific performance metrics must be clearly defined, and the necessity for these metrics must also be clarified.
2. Benchmarking system developers should support fine-grained result generation. Since the metric definition varies for each blockchain, publishing all variations of a metric in the results will be helpful for better understanding. In most cases, simple mathematical calculations can provide more fine-grained results. For example, the results from the Caliper benchmarking system display only the "*success throughput*" and not the "*commit throughput*" even though both can be derived from the available results [9].

4 Case Study

In this section, we analyze five different benchmarking systems that support permissioned blockchains [5,8,22,36,40] as well as the corresponding five benchmarking studies conducted using these systems [9,28,33,57,65]. Our discussion is mainly based on the benchmarking studies as this is representative of how the benchmarking system is used in practice. Table 1 summarizes the integrated blockchain systems, available workloads, and published performance metrics for each benchmarking system. We intend to identify the limitations of the current benchmarking systems through this case study which can help develop a more comprehensive system.

Scope. We observe that none of the benchmarking systems currently support all four of the most commonly used permissioned blockchains (Fabric, Corda, Multichain and Quorum). One of the main reasons for benchmarking is for clients to choose the appropriate blockchains based on their requirements. Therefore, a benchmarking system must support at least the most popular blockchain choices. However, the engineering challenge behind implementing such a comprehensive benchmarking system is immense. Alternatively, providing documentation that accurately details the exact procedure to integrate any new blockchain into an existing benchmarking system would be beneficial. Diablo, Gromit, and BCT-Mark provide short documentation or discussions on integrating new blockchains into their benchmarking system [23,57,65]. Caliper provides extensive documentation that details the steps required to implement a connector to integrate a new blockchain [75]. This includes the requirements of the connector, implementation, binding, and integration, as well as instructions on how to document

Table 1. Blockchain Benchmarking Systems

Benchmarking Systems	**Supported blockchains (permissioned underlined)**	**Supported Workloads**	**Performance Metrics**
Blockbench [28]	Ethereum [74], Fabric [2], Parity [58], Quorum [60]	YCSB, smallbank, etherId, doubler, wavesPresale, doNothing, analytics, IOHeavy, CPUHeavy [5]	success throughput, average latency
HyperledgerLab [9], Caliper [40]	Fabric, Ethereum, Besu [39]	simple asset transfer, smallbank, fabcar, synthetic generator, electronic health records, digital music management, e-voting, supply chain management [41, 43]	commit throughput, success throughput, average latency
Diablo [33]	Algorand [29], Avalanche [63], Ethereum, Diem [3], Solana [68], Quorum, RedBelly [21, 69]	exchange DApp, gaming DApp, webservice DApp, mobility service DApp, video sharing DApp [22]	throughput, average latency, proportion of committed transactions, peak transaction throughput, latency distribution over time
Gromit [57]	Ethereum, Algorand, BitShares [66], Diem, Fabric, Stellar [48], Avalanche	simple asset transfer [36]	peak transaction throughput, average latency
BCTMark [65]	Ethereum, Clique [13], Fabric	synthetic generator, history-based , sorting algorithms [8]	CPU usage, HDD usage, memory consumption, gas cost

the newly developed connector for future users. Despite the well-defined documentation, there has been little effort from the community to integrate more blockchains into Caliper.

System Configuration. The existing benchmarking systems support the evaluation of the different blockchains on scaling hardware configurations. Diablo, Gromit, and HyperledgerLab emulate geo-distribution. However, system configuration is not extensively evaluated in the corresponding benchmarking studies. The number of hardware nodes and, correspondingly, the number of peers in a system are scaled and evaluated but the peer configurations are kept constant. Currently, system configuration is considered independent from the benchmarking systems and is left to the client's responsibility. Providing automated testbed setups for the supported blockchains can ease the benchmarking effort with varying system configurations. The HyperledgerLab benchmarking system includes such an automated testbed and therefore can evaluate the effect of endorser and database configurations, but it is limited to Fabric.

Parameter Tuning. In the studies we examined, system parameters are mostly kept with the default value used in whichever blockchain is being tested. Blockbench tunes the *difficulty* variable for Ethereum to limit miners from diverging [28]. HyperledgerLab evaluates the effect of system parameters such as *block size* and *endorsement policy* but is limited to Fabric [9]. Tuning the parameters of individual blockchains to ensure the fair comparison of the best performance of all the systems under test is a massive challenge due to the large number of parameters involved. Currently, we identified some of the prominent parameters for the different blockchains discussed in this paper by manually parsing through the multiple documentations and configuration files [15, 17, 18, 26, 30, 32, 38, 51]. Blockchain developers must provide more intuitive documentation regarding the performance tuning of their specific blockchain implementation. Consequently, benchmarking systems could automate parameter tuning to ease the benchmarking process.

Workloads and Use Cases. Workloads are well investigated by the existing benchmarking systems, and the supported workloads for each are listed in Table 1. Diablo extracts the workload trace from five real centralized applications and designs corresponding decentralized applications (DApps) to provide a realistic blockchain-specific benchmarking scenario. Blockbench provides popular database benchmarking workloads such as YCSB and small bank, which provides a good understanding of the contrast between blockchains and databases. Blockbench also supports microbenchmarks such as IO-heavy and CPU-heavy, while HyperledgerLab provides synthetic workloads such as read-heavy, update-heavy, or skewed keys. All the existing benchmarking systems also support workloads at different transaction rates. Overall, the workloads supported by the existing benchmarking systems cover many practical and synthetic use cases, ensuring a comprehensive blockchain evaluation. There are also many other blockchain specific workloads available in the literature [9,10,56]. However, developing or extending a benchmarking system to include this extensive set of workloads would be advantageous. The evaluation results of the existing benchmarking systems are well explored and discussed in their corresponding papers. For example, Nasrulin et al. [57] highlight six different findings that summarize the performance of the compared blockchains. However, relating the evaluation results to the implementation specifics of the blockchains and the intended use cases would be helpful for a client trying to choose the ideal blockchain. Gramoli et al. [33] observe that Diem and Avalanche do not support challenging hardware configurations but also point out that such configurations may not be the intended use case for these blockchains. They also highlight that blockchains with eventual consistency scale better, providing a client who requires immediate consistency with realistic expectations. The intended use cases of a specific blockchain and its implementation specifics, such as its consistency and fault tolerance models, need to be effectively explored while discussing benchmarking results.

Performance Metrics. The existing benchmarking studies evaluate a wide range of performance metrics. Gromit focuses on the peak transaction throughput, the maximum throughput supported by a system before it hangs. Diablo measures the average throughput and a throughput time series, including the peak throughput. BCTMark focuses more on system metrics such as CPU and memory usage. Blockbench measures the success throughput, while HyperledgerLab evaluates the committed throughput, including failed transactions. The importance and reasoning of each of the metrics are well-defined in the benchmarking studies. However, a single benchmarking system that provides a comprehensive set of all the different metrics would be valuable.

5 Related Work

The literature proposes various benchmarking systems and corresponding benchmarking studies for permissioned blockchains, which we have analyzed in our case study. Dinh et al. developed the first benchmarking system for permissioned blockchains with a precise definition for the different abstraction layers [28]. HyperledgerLab [9], which uses the Caliper benchmarking system [40], implemented an automated blockchain (Fabric) network deployment tool to simplify benchmarking experiments. Saingre et al. proposed a blockchain benchmarking system that adheres to the six criteria for a good benchmark [65, 67]. Nasrulin et al. investigated the popular consensus protocols and benchmarked representative blockchain systems for each [57]. Gramoli et al. implemented realistic distributed applications to evaluate multiple blockchain systems' performance uniformly [33]. The existing publications focus on developing a benchmarking system, while our work highlights general benchmarking guidelines for the blockchain community, which includes both blockchain and benchmarking system developers. Benchmarks and benchmarking systems are well-established research areas in the database community [4, 6, 34, 61]. However, despite the similarities, the implementation and application differences demand a separate discussion for benchmarking blockchains [64].

6 Conclusion

We analyzed five permissioned blockchains to define specific problem statements regarding four main aspects of benchmarking blockchains. We provide examples from each of the chosen platforms to clarify the problem statements. Further, we discuss a general methodology to tackle each problem statement, highlighting the need for contributions from the developers and users of blockchains and benchmarking systems. We then conducted a case study of five different permissioned blockchain benchmarking systems and the affiliated benchmarking studies based on our problem statements. We emphasize the current limitations of these systems, which can help improve the state-of-the-art. Given the implementation differences between blockchains and the numerous components, configuration parameters, and metrics specific to each blockchain, one main conclusion from our work is the need for blockchain developers to actively engage in the benchmarking space. We urge blockchain developers to quantitatively identify and define system-specific factors such as the top parameters to tune, the ideal system setup for a fixed hardware configuration or cost, targeted use cases, and performance metrics that can ease the process of benchmarking blockchains.

Acknowledgement. This work is funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 392214008, and by the Bavarian Cooperative Research Program of the Free State of Bavaria - DIK-2002-0013//DIK0114/02.

References

1. Amsden, Z.: The libra blockchain. url: https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf (2020)
2. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. pp. 1–15 (2018)
3. Bano, S., Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., Malkhi, D., Naor, O., Perelman, D., et al.: State machine replication in the libra blockchain. Avalaible at: https://developers. libra. org/docs/state-machine-replication-paper (Consulted on December 19, 2020) (2020)
4. Bermbach, D., Wittern, E., Tai, S.: Cloud service benchmarking. Springer (2017)
5. Blockbench. https://github.com/ooibc88/blockbench (2020), [Online; accessed 14-August-2023]
6. Brent, L., Fekete, A.: A versatile framework for painless benchmarking of database management systems. In: Chang, L., Gan, J., Cao, X. (eds.) Databases Theory and Applications. Lecture Notes in Computer Science, vol 11393. pp. 45–56. Springer International Publishing, Cham (2019)
7. Brown, R.G., Carlyle, J., Grigg, I., Hearn, M.: Corda: an introduction. R3 CEV, August **1**(15), 14 (2016)
8. Btcmark. https://gitlab.inria.fr/dsaingre/bctmark (2020), [Online; accessed 14-August-2023]
9. Chacko, J.A., Mayer, R., Jacobsen, H.A.: Why do my blockchain transactions fail? a study of hyperledger fabric. In: Proceedings of the 2021 International Conference on Management of Data. pp. 221–234. SIGMOD/PODS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3448016.3452823, https://doi.org/10.1145/3448016.3452823
10. Chacko, J.A., Mayer, R., Jacobsen, H.A.: How to optimize my blockchain? a multi-level recommendation approach. Proc. ACM Manag. Data **1**(1) (may 2023). https://doi.org/10.1145/3588704, https://doi.org/10.1145/3588704
11. Chainhammer. https://github.com/drandreaskrueger/chainhammer (2020), [Online; accessed 14-August-2023]
12. Chung, G., Desrosiers, L., Gupta, M., Sutton, A., Venkatadri, K., Wong, O., Zugic, G.: Performance tuning and scaling enterprise blockchain applications. arXiv preprint arXiv:1912.11456 (2019)
13. Clique proof-of-authority consensus protocol. https://eips.ethereum.org/EIPS/eip-225 (2020), [Online; accessed 14-August-2023]
14. Configure bootnodes. https://consensys.net/docs/goquorum/en/latest/configure-and-manage/configure/bootnodes/ (2020), [Online; accessed 14-August-2023]
15. Configure static nodes. https://consensys.net/docs/goquorum/en/latest/configure-and-manage/configure/static-nodes/ (2020), [Online; accessed 14-August-2023]
16. Consensus protocols. https://docs.goquorum.consensys.net/concepts/consensus (2020), [Online; accessed 14-August-2023]
17. Corda configurations. https://docs.r3.com/en/platform/corda/4.8/enterprise/node/setup/corda-configuration-fields.html (2020), [Online; accessed 14-August-2023]
18. Corda key concepts. https://docs.r3.com/en/platform/corda/4.10/community/key-concepts.html (2020), [Online; accessed 14-August-2023]

19. Corda notaries. https://docs.r3.com/en/platform/corda/4.10/community/key-concepts-notaries.html (2020), [Online; accessed 14-August-2023]
20. Corda use case directory. https://r3.com/products/use-case-directory-all/ (2023), [Online; accessed 14-August-2023]
21. Crain, T., Natoli, C., Gramoli, V.: Red belly: a secure, fair and scalable open blockchain. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 466–483. IEEE (2021)
22. Diablo blockchain benchmark suite. https://diablobench.github.io/ (2020), [Online; accessed 14-August-2023]
23. Diablo blockchain benchmark suite. https://diablobench.github.io/blockchain-howto (2020), [Online; accessed 14-August-2023]
24. Diem consensus configurations. https://github.com/diem/diem/blob/latest/config/src/config/consensus_config.rs (2020), [Online; accessed 14-August-2023]
25. Diem mempool configurations. https://github.com/diem/diem/blob/latest/config/src/config/mempool_config.rs (2020), [Online; accessed 14-August-2023]
26. Diem validator nodes. https://developers.diem.com/docs/basics/basics-validator-nodes (2020), [Online; accessed 14-August-2023]
27. Diem white paper. https://developers.diem.com/docs/technical-papers/the-diem-blockchain-paper/ (2020), [Online; accessed 14-August-2023]
28. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: A framework for analyzing private blockchains. In: Proceedings of the 2017 ACM international conference on management of data. pp. 1085–1100 (2017)
29. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 51–68. SOSP '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132747.3132757, https://doi.org/10.1145/3132747.3132757
30. Go ethereum. https://geth.ethereum.org/docs/interface/command-line-options (2020), [Online; accessed 14-August-2023]
31. Goquorum configuration file. https://consensys.net/docs/goquorum/en/latest/configure-and-manage/configure/use-configuration-file/ (2020), [Online; accessed 14-August-2023]
32. Goquorum qlight. https://consensys.net/docs/goquorum/en/latest/concepts/qlight-node/ (2020), [Online; accessed 14-August-2023]
33. Gramoli, V., Guerraoui, R., Lebedev, A., Natoli, C., Voron, G.: Diablo: A benchmark suite for blockchains. In: 18th ACM European Conference on Computer Systems (EuroSys). p. to appear (2023)
34. Gray, J.: Benchmark handbook: for database and transaction processing systems. Morgan Kaufmann Publishers Inc. (1992)
35. Greenspan, G., et al.: Multichain private blockchain-white paper. url: http://www. multichain. com/download/MultiChain-White-Paper. pdf pp. 57–60 (2015)
36. Gromit blockchain benchmarking tool. https://github.com/grimadas/gromit (2020), [Online; accessed 14-August-2023]
37. Hearn, M., Brown, R.G.: Corda: A distributed ledger. Corda Technical White Paper **2016** (2016)
38. How fabric networks are structured. https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html (2020), [Online; accessed 14-August-2023]
39. Hyperledger besu. https://www.hyperledger.org/use/besu (2020), [Online; accessed 14-August-2023]

40. https://hyperledger.github.io/caliper/ (2020), [Online; accessed 14-August-2023]
41. Hyperledger caliper benchmarks. https://github.com/hyperledger/caliper-benchmarks (2020), [Online; accessed 14-August-2023]
42. Hyperledger foundation case studies. https://www.hyperledger.org/learn/case-studies (2023), [Online; accessed 14-August-2023]
43. Hyperledgerlab ii. https://github.com/MSRG/HyperLedgerLab-2.0 (2020), [Online; accessed 14-August-2023]
44. Klenik, A., Kocsis, I.: Porting a benchmark with a classic workload to blockchain: Tpc-c on hyperledger fabric. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. pp. 290–298 (2022)
45. Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: Proceedings of the 25th USENIX Conference on Security Symposium. p. 279–296. SEC'16, USENIX Association, USA (2016)
46. Li, M., Wang, Y., Ma, S., Liu, C., Huo, D., Wang, Y., Xu, Z.: Auto-tuning with reinforcement learning for permissioned blockchain systems. Proc. VLDB Endow. **16**(5), 1000–1012 (february 2023). https://doi.org/10.14778/3579075.3579076, https://doi:10.14778/3579075.3579076
47. Liu, M., Yu, F.R., Teng, Y., Leung, V.C.M., Song, M.: Performance optimization for blockchain-enabled industrial internet of things (iiot) systems: A deep reinforcement learning approach. IEEE Transactions on Industrial Informatics **15**(6), 3559–3570 (2019). https://doi.org/10.1109/TII.2019.2897805
48. Lokhava, M., Losa, G., Mazières, D., Hoare, G., Barry, N., Gafni, E., Jove, J., Malinowsky, R., McCaleb, J.: Fast and secure global payments with stellar. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. pp. 80–96. SOSP '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3341301.3359636, https://doi.org/10.1145/3341301.3359636
49. Mazzoni, M., Corradi, A., Di Nicola, V.: Performance evaluation of permissioned blockchains for financial applications: The consensys quorum case study. Blockchain: Research and Applications **3**(1), 100026 (2022). https://doi.org/https://doi.org/10.1016/j.bcra.2021.100026
50. Morgan, J.: Quorum whitepaper. New York: JP Morgan Chase (2016)
51. Multichain configurations. https://www.multichain.com/developers/blockchain-parameters/ (2020), [Online; accessed 14-August-2023]
52. Multichain data streams. https://www.multichain.com/developers/data-streams/ (2020), [Online; accessed 14-August-2023]
53. Multichain performance optimization. https://www.multichain.com/developers/performance-optimization/ (2020), [Online; accessed 14-August-2023]
54. Multichain product partners. https://www.multichain.com/product-partners/ (2023), [Online; accessed 14-August-2023]
55. Multichain white paper. https://www.multichain.com/download/MultiChain-White-Paper.pdf (2020), [Online; accessed 14-August-2023]
56. Nasirifard, P., Mayer, R., Jacobsen, H.A.: Fabriccrdt: A conflict-free replicated datatypes approach to permissioned blockchains. In: Proceedings of the 20th International Middleware Conference. p. 110–122. Middleware '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3361525.3361540, https://doi.org/10.1145/3361525.3361540

57. Nasrulin, B., De Vos, M., Ishmaev, G., Pouwelse, J.: Gromit: Benchmarking the performance and scalability of blockchain systems. In: 2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). pp. 56–63. IEEE (2022)
58. Parity: Blockchain infrastructure for the decentralised web. https://www.parity.io/ (2020), [Online; accessed 14-August-2023]
59. Quorum blockchain in action. https://consensys.net/quorum/enterprise/ (2023), [Online; accessed 14-August-2023]
60. Quorum white paper. https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf (2020), [Online; accessed 14-August-2023]
61. Raasveldt, M., Holanda, P., Gubner, T., Mühleisen, H.: Fair benchmarking considered difficult: Common pitfalls in database performance testing. In: Proceedings of the Workshop on Testing Database Systems. DBTest'18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3209950.3209955, https://doi.org/10.1145/3209950.3209955
62. Rauchs, M., Blandin, A., Bear, K., McKeon, S.B.: 2nd global enterprise blockchain benchmarking study. Available at SSRN 3461765 (2019)
63. Rocket, T., Yin, M., Sekniqi, K., van Renesse, R., Sirer, E.G.: Scalable and probabilistic leaderless bft consensus through metastability (2020)
64. Ruan, P., Dinh, T.T.A., Loghin, D., Zhang, M., Chen, G., Lin, Q., Ooi, B.C.: Blockchains vs. distributed databases: Dichotomy and fusion. In: Proceedings of the 2021 International Conference on Management of Data. p. 1504–1517. SIGMOD '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3448016.3452789
65. Saingre, D., Ledoux, T., Menaud, J.M.: Bctmark: a framework for benchmarking blockchain technologies. In: 2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA). pp. 1–8. IEEE (2020)
66. Schuh, F., Larimer, D.: Bitshares 2.0: General overview (2017)
67. Smaalders, B.: Performance anti-patterns: Want your apps to run faster? here's what not to do. Queue **4**(1) (feb 2006). https://doi.org/10.1145/1117389.1117403
68. Solana: A new architecture for a high performance blockchain v0.8.13. https://solana.com/solana-whitepaper.pdf (2020), [Online; accessed 14-August-2023]
69. Tennakoon, D., Gramoli, V.: Smart red belly blockchain: Enhanced transaction management for decentralized applications. arXiv preprint arXiv:2207.05971 (2022)
70. Thakkar, P., Nathan, S., Viswanathan, B.: 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). pp. 264–276 (Sep 2018). https://doi.org/10.1109/MASCOTS.2018.00034
71. Tran, N.K., Babar, M.A., Walters, A.: A framework for automating deployment and evaluation of blockchain networks. Journal of Network and Computer Applications **206**, 103460 (2022). https://doi.org/https://doi.org/10.1016/j.jnca.2022.103460
72. Types of transactions. https://developers.diem.com/docs/transactions/txns-types/ (2020), [Online; accessed 14-August-2023]
73. Valenta, M., Sandner, P.G.: Comparison of ethereum, hyperledger fabric and corda (2017)
74. Wood, D.D.: Ethereum: A secure decentralised generalised transaction ledger (2014)

75. Writing connectors. https://hyperledger.github.io/caliper/v0.5.0/writing-connectors/ (2020), [Online; accessed 14-August-2023]

76. Zhang, J., Gao, J., Wu, Z., Yan, W., Wo, Q., Li, Q., Chen, Z.: Performance analysis of the libra blockchain: An experimental study. In: 2019 2nd International Conference on Hot Information-Centric Networking (HotICN). pp. 77–83. IEEE (2019)

Licence to Publish
Proceedings Papers

SPRINGER NATURE

Licensee	Springer Nature Switzerland AG	(the 'Licensee')
Title of the Proceedings Volume/Edited Book or Conference Name:	TPCTC 2023	(the 'Volume')
Volume Editor(s) Name(s):	Raghunath Nambiar, Meikel Poess	
Proposed Title of the Contribution:	A Comprehensive Study on Benchmarking Permissioned Blockchains	(the 'Contribution')
Series: The Contribution may be published in the following series	A Springer Nature Computer Science book series (CCIS, LNAI, LNBI, LNBIP or LNCS)	
Author(s) Full Name(s):	Jeeta Ann Chacko, Ruben Mayer, Alan Fekete, Vincent Gramoli, Hans-Arno Jacobsen	(the 'Author')

When Author is more than one person the expression "Author" as used in this Agreement will apply collectively unless otherwise indicated.

Corresponding Author Name:	Jeeta Ann Chacko	
Instructions for Authors	https://resource-cms.springernature.com/springer-cms/rest/v1/content/19242230/data/	(the 'Instructions for Authors')

1 Grant of Rights

a) For good and valuable consideration, the Author hereby grants to the Licensee the perpetual, exclusive, world-wide, assignable, sublicensable and unlimited right to: publish, reproduce, copy, distribute, communicate, display publicly, sell, rent and/or otherwise make available the contribution identified above, including any supplementary information and graphic elements therein (e.g. illustrations, charts, moving images) (the 'Contribution') in any language, in any versions or editions in any and all forms and/or media of expression (including without limitation in connection with any and all end-user devices), whether now known or developed in the future. Without limitation, the above grant includes: (i) the right to edit, alter, adapt, adjust and prepare derivative works; (ii) all advertising and marketing rights including without limitation in relation to social media; (iii) rights for any training, educational and/or instructional purposes; (iv) the right to add and/or remove links or combinations with other media/works; and (v) the right to create, use and/or license and/or sublicense content data or metadata of any kind in relation to the Contribution (including abstracts and summaries) without restriction. The above rights are granted in relation to the Contribution as a whole or any part and with or in relation to any other works.

b) Without limiting the rights granted above, Licensee is granted the rights to use the Contribution for the purposes of analysis, testing, and development of publishing- and research-related workflows, systems, products, projects, and services; to confidentially share the Contribution with select third parties to do the same; and to retain and store the Contribution and any associated correspondence/files/forms to maintain the historical record, and to facilitate research integrity investigations. The grant of rights set forth in this clause (b) is irrevocable.

c) If the Licensee elects not to publish the Contribution for any reason, all publishing rights under this Agreement as set forth in clause 1a above will revert to the Author.

2 Copyright

Ownership of copyright in the Contribution will be vested in the name of the Author. When reproducing the Contribution or extracts from it, the Author will acknowledge and reference first publication in the Volume.

3 Use of Contribution Versions

a) For purposes of this Agreement: (i) references to the "Contribution" include all versions of the Contribution; (ii) "Submitted Manuscript" means the version of the Contribution as first submitted by the Author prior to peer review; (iii) "Accepted Manuscript" means the version of the Contribution accepted for publication, but prior to copy-editing and typesetting; and (iv) "Version of Record" means the version of the Contribution published by the Licensee, after copy-editing and typesetting. Rights to all versions of the Manuscript are granted on an exclusive basis, except for the Submitted Manuscript, to which rights are granted on a non-exclusive basis.

b) The Author may make the Submitted Manuscript available at any time and under any terms (including, but not limited to, under a CC BY licence), at the Author's discretion. Once the Contribution has been published, the Author will include an acknowledgement and provide a link to the Version of Record on the publisher's website: "This preprint has not undergone peer review (when applicable) or any post-submission improvements or corrections. The Version of Record of this contribution is published in [insert volume title], and is available online at https://doi.org/[insert DOI]".

c) The Licensee grants to the Author (i) the right to make the Accepted Manuscript available on their own personal, self-maintained website immediately on acceptance, (ii) the right to make the Accepted Manuscript available for public release on any of the following twelve (12) months after first publication (the "Embargo Period"): their employer's internal website; their institutional and/or funder repositories. Accepted Manuscripts may be deposited in such repositories immediately upon acceptance, provided they are not made publicly available until after the Embargo Period.
The rights granted to the Author with respect to the Accepted Manuscript are subject to the conditions that (i) the Accepted Manuscript is not enhanced or substantially reformatted by the Author or any third party, and (ii) the Author includes on the Accepted Manuscript an acknowledgement in the following form, together with a link to the published version on the publisher's website: "This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/[insert DOI]. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms".
Under no circumstances may an Accepted Manuscript be shared or distributed under a Creative Commons or other form of open access licence.
Any use of the Accepted Manuscript not expressly permitted under this subclause (c) is

subject to the Licensee's prior consent.

d) The Licensee grants to Author the following non-exclusive rights to the Version of Record, provided that, when reproducing the Version of Record or extracts from it, the Author acknowledges and references first publication in the Volume according to current citation standards. As a minimum, the acknowledgement must state: "First published in [Volume, page number, year] by Springer Nature".

i. to reuse graphic elements created by the Author and contained in the Contribution, in presentations and other works created by them;

ii. the Author and any academic institution where they work at the time may reproduce the Contribution for the purpose of course teaching (but not for inclusion in course pack material for onward sale by libraries and institutions);

iii. to reuse the Version of Record or any part in a thesis written by the same Author, and to make a copy of that thesis available in a repository of the Author(s)' awarding academic institution, or other repository required by the awarding academic institution. An acknowledgement should be included in the citation: "Reproduced with permission from Springer Nature";

iv. to reproduce, or to allow a third party to reproduce the Contribution, in whole or in part, in any other type of work (other than thesis) written by the Author for distribution by a publisher after an embargo period of 12 months; and

v. to publish an expanded version of their Contribution provided the expanded version (i) includes at least 30% new material (ii) includes an express statement specifying the incremental change in the expanded version (e.g., new results, better description of materials, etc.).

4 Warranties & Representations

Author warrants and represents that:

a)

i. the Author is the sole copyright owner or has been authorised by any additional copyright owner(s) to grant the rights defined in clause 1,

ii. the Contribution does not infringe any intellectual property rights (including without limitation copyright, database rights or trade mark rights) or other third party rights and no licence from or payments to a third party are required to publish the Contribution,

iii. the Contribution has not been previously published or licensed, nor has the Author committed to licensing any version of the Contribution under a licence inconsistent with the terms of this Agreement,

iv. if the Contribution contains materials from other sources (e.g. illustrations, tables, text quotations), Author has obtained written permissions to the extent necessary from the copyright holder(s), to license to the Licensee the same rights as set out in clause 1 but on a non-exclusive basis and without the right to use any graphic elements on a stand-alone basis and has cited any such materials correctly;

b) all of the facts contained in the Contribution are according to the current body of research true and accurate;

c) nothing in the Contribution is obscene, defamatory, violates any right of privacy or publicity, infringes any other human, personal or other rights of any person or entity or is otherwise unlawful and that informed consent to publish has been obtained for any research participants;

d) nothing in the Contribution infringes any duty of confidentiality owed to any third party or violates any contract, express or implied, of the Author;

e) all institutional, governmental, and/or other approvals which may be required in connection with the research reflected in the Contribution have been obtained and continue in effect;

f) all statements and declarations made by the Author in connection with the Contribution are true and correct;

g) the signatory who has signed this Agreement has full right, power and authority to enter into this Agreement on behalf of all of the Authors; and

h) the Author complies in full with: i. all instructions and policies in the Instructions for Authors, ii. the Licensee's ethics rules (available at https://www.springernature.com/gp/authors/book-authors-code-of-conduct), as may be updated by the Licensee at any time in its sole discretion.

5 Cooperation

a) The Author will cooperate fully with the Licensee in relation to any legal action that might arise from the publication of the Contribution, and the Author will give the Licensee access at reasonable times to any relevant accounts, documents and records within the power or control of the Author. The Author agrees that any Licensee affiliate through which the Licensee exercises any rights or performs any obligations under this Agreement is intended to have the benefit of and will have the right to enforce the terms of this Agreement.

b) Author authorises the Licensee to take such steps as it considers necessary at its own expense in the Author's name(s) and on their behalf if the Licensee believes that a third party is infringing or is likely to infringe copyright in the Contribution including but not limited to initiating legal proceedings.

6 Author List

Changes of authorship, including, but not limited to, changes in the corresponding author or the sequence of authors, are not permitted after acceptance of a manuscript.

7 Post Publication Actions

The Author agrees that the Licensee may remove or retract the Contribution or publish a correction or other notice in relation to the Contribution if the Licensee determines that such actions are appropriate from an editorial, research integrity, or legal perspective.

8 Controlling Terms

The terms of this Agreement will supersede any other terms that the Author or any third party may assert apply to any version of the Contribution.

9 Governing Law

This Agreement shall be governed by, and shall be construed in accordance with, the laws of Switzerland. The courts of Zug, Switzerland shall have the exclusive jurisdiction.

Signed for and on behalf of the Author	Print Name:	Date:
	JEETA ANN CHACKO	14TH AUGUST 2023

Address:	Technical University of Munich, Boltzmannstr. 3, 85748 Garching, Germany
Email:	chacko@in.tum.de

Appendix E

Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric

Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric

Jeeta Ann Chacko
chacko@in.tum.de
Technical University of Munich

Nino Richter
nino.richter@tum.de
Technical University of Munich

Ruben Mayer
ruben.mayer@uni-bayreuth.de
University of Bayreuth

Hans-Arno Jacobsen
jacobsen@eecg.toronto.edu
University of Toronto

Abstract

Hyperledger Fabric is a popular permissioned blockchain that follows an optimistic concurrency control model, which often leads to transaction dependency conflicts. We developed a web application that extracts data from a live Fabric network and generates transaction dependency graphs. Apart from visually understanding the transaction dependency distribution, the user also gets additional information regarding each transaction, such as the validation status, dependent transactions, smart contract function, accessed keys, endorsers and clients. Further, our tool also checks the serializability of the generated graph and identifies the number of transactions to abort to achieve serializability. We demonstrate how our tool can be used with multiple realistic workloads to identify performance optimization opportunities. We also highlight scenarios where our tool can be used to identify smart contract improvements. Our tool is demonstrated with multiple realistic workloads.

CCS Concepts

• **Information systems → Data management systems**.

Keywords

blockchains, transaction dependency, hyperledger fabric

ACM Reference Format:
Jeeta Ann Chacko, Nino Richter, Ruben Mayer, and Hans-Arno Jacobsen. 2023. Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric. In *24th International Middleware Conference Demos, Posters and Doctoral Symposium (Middleware Demos, Posters and Doctoral Symposium'23), December 11–15, 2023, Bologna, Italy*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3626564.3629098

1 Introduction

Hyperledger Fabric (a.k.a. Fabric) [3] is one of the most commonly used permissioned blockchains [4]. Fabric uses an optimistic concurrency control model that leads to transaction dependency conflicts which can be quite frequent in certain scenarios [1]. Performance optimization of Fabric is an active area of research with many of the optimization strategies focusing purely on reducing transaction dependency failures [5, 6]. However, it has been quantitatively shown that these optimizations cannot be blindly applied and require an in-depth understanding about the blockchain application and the corresponding workload [1].

To help users assess the effectiveness of various Fabric optimizations, we developed `Fabric-Visualizer`, a web application that can parse the distributed ledger of a Fabric network and provides transaction failure analysis which help users better understand their workload. It generates a transaction conflict graph that also shows additional information for each transaction such as the validation status, dependent transactions, smart contract function, accessed keys, endorsers and clients. This helps the user to exactly identify the factors that caused a transaction failure and aids in smart contract optimization. Further, the tool also checks the serializability of the transactions and identifies the transactions that need to be aborted to achieve serializability. This helps the user in their choice of optimization strategies. We demonstrate multiple scenarios using four different realistic workloads that highlight the utility of the tool (Section 3). `Fabric-Visualizer` provides a user-friendly approach to understanding your blockchain workload which can assist in choosing the right optimization strategy or improving your smart contract.

2 System Overview

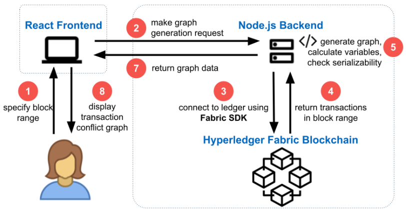

React Frontend
2
make graph generation request
Node.js Backend
</> generate graph, calculate variables, check serializability
5
7
return graph data
1
specify block range
8
display transaction conflict graph
3
connect to ledger using Fabric SDK
4
return transactions in block range
Hyperledger Fabric Blockchain

Figure 1: System Overview

The system overview of `Fabric-Visualizer` is illustrated in Figure 1. Users can specify the range of blocks they wish to analyze in the application frontend and send a request. The backend of our application connects to a Fabric network and retrieves the blockchain data. A transaction dependency graph is then generated along with the measurement of multiple failure-related information. The graph is then tested for serializability. If a cyclic graph is detected, then the transactions that need to be aborted to serialize the graph are also identified. All the derived information is displayed on the frontend.

Middleware Demos, Posters and Doctoral Symposium'23 , December 11–15, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0429-1/23/12…$15.00
https://doi.org/10.1145/3626564.3629098

3 Demonstration Scenarios

The implementation of the tool is complete and our git repository[1] as well as a video demonstrating the tool[2] is publicly accessible. Since real logs of permissioned blockchains are rarely available publicly, we rely on realistic workloads generated for popular blockchain use cases such as supply chain management (SCM), electronic health record management (EHR), digital rights management (DRM), and digital voting (DV) which were developed in our previous work [1]. Our tool can be used to better understand these workloads and derive useful insights.

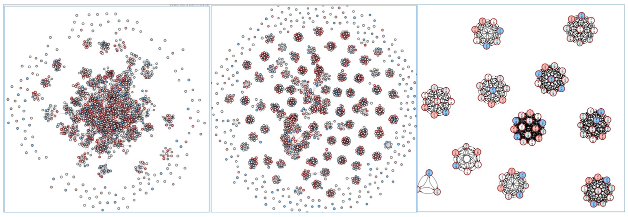
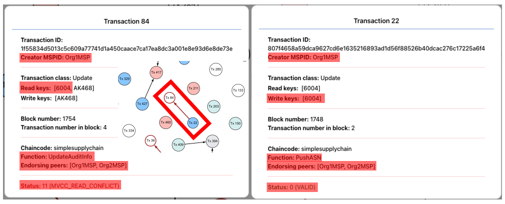


Figure 2: Conflict graphs with the EHR (left), DRM (middle) and DV (right) workloads.

Figure 2 shows the conflict graphs generated for the EHR and DRM workloads. We can observe that there is a prominently large, interconnected cluster for the EHR conflict graph, while there are several smaller interconnected clusters for DRM. This indicates the likelihood that fewer transactions might need to be aborted for serializability in the EHR workload since a large proportion of transactions are interconnected. For the DRM workload, since every small interconnected cluster needs to be made acyclic, the number of transaction abortions might be higher. This intuitive observation can also be confirmed quantitatively using our tool, which reports that there are 319 failed transactions in the EHR workload and serializability is possible if 151 transactions are aborted. For the DRM workload, which has 336 failed transactions, 306 transaction abortions are required to achieve serializability. This example demonstrates how our tool can help the user assessing the effectiveness of popular optimization strategies such as transaction reordering and early aborts [5, 6]. Compared to DRM, EHR has a higher ratio between failed transactions and number of transaction aborts to achieve serializability. Hence, it is more promising to optimize the EHR scenario than the DRM scenario using the discussed optimization strategies.

Figure 2 also shows the conflict graph generated for the DV workload. Like DRM, the graph shows multiple tightly interconnected clusters. Our tool also indicates a large number of transaction failures and that the transactions are not serializable. When we further explored the keys accessed by the transactions, we observe that all transactions in each interconnected cluster in the graph is dependent on the same key i.e., the *partyID*. This implies that each cluster is composed of the transactions that voted for the same party. The same scenario has been analyzed in our recent paper [2] where data model alteration was detected as an effective optimization strategy.

Figure 3 shows multiple screenshots of the transaction details for two dependent transactions from the conflict graph generated for the SCM workload. We observe that transactions numbered 22 and 84 have a dependency that results in the failure (MVCC read conflict) of one of them (transaction 84). Our tool also shows the exact smart contract functions and keys that caused this transaction dependency conflict. We can then analyze these specific functions in the smart contract to identify possible optimizations. Detecting the smart contract functions and keys responsible for transaction conflicts can lead to identifying effective optimization strategies such as smart contract partitioning, data model alteration, and activity pruning [2]. Further, our tool also shows the client that generated a transaction and the peers that endorsed it (*Creator MSPID* and *Endorsing peers* in Figure 3) which can aid in the identification of optimizations such as client-based reordering, client resource boost, endorser restructuring, and transaction prioritization [2, 7].



Transaction 84

Transaction ID:
1f55834d5013c5c609a77741d1a450caace7ca17ea8dc3a001e8e93d6e8de73e
Creator MSPID: Org1MSP

Transaction class: Update
Read keys: [6004, AK468]
Write keys: [AK468]

Block number: 1754
Transaction number in block: 4

Chaincode: simplesupplychain
Function: UpdateAuditInfo
Endorsing peers: [Org1MSP, Org2MSP]

Status: 11 (MVCC_READ_CONFLICT)

Transaction 22

Transaction ID:
807f4658a59dca9627cd6e1635216893ad1d56f88526b40dcac276c17225a6f4
Creator MSPID: Org1MSP

Transaction class: Update
Read keys: [6004]
Write keys: [6004]

Block number: 1748
Transaction number in block: 2

Chaincode: simplesupplychain
Function: PushASN
Endorsing peers: [Org1MSP, Org2MSP]

Status: 0 (VALID)

Figure 3: Details of two dependent transactions

4 Conclusions

We developed Fabric-Visualizer, a user-friendly web application that helps to analyze transaction dependencies in Fabric workloads. The implementation of our tool is complete and it is publicly available. We have demonstrated multiple scenarios to highlight the utility of the tool for a Fabric user.

Acknowledgments

This work is funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 392214008, and by the Bavarian Cooperative Research Program of the Free State of Bavaria - DIK-2002-0013//DIK0114/02.

References

[1] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. Association for Computing Machinery.

[2] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2023. How To Optimize My Blockchain? A Multi-Level Recommendation Approach. *Proc. ACM Manag. Data* 1, 1, Article 24 (may 2023). https://doi.org/10.1145/3588704

[3] Androulaki et al. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. ACM, Article 30, 30:1–30:15 pages.

[4] Michel Rauchs, Apolline Blandin, Keith Bear, and Stephen B McKeon. 2019. 2nd global enterprise blockchain benchmarking study. *Available at SSRN 3461765* (2019).

[5] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A Transactional Perspective on Execute-Order-Validate Blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. https://doi.org/10.1145/3318464.3389693

[6] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the Lines Between Blockchains and Database Systems: The Case of Hyperledger Fabric. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. ACM, New York, NY, USA, 105–122. https://doi.org/10.1145/3299869.3319883

[7] Shenbin Zhang, Ence Zhou, Bingfeng Pi, Jun Sun, Kazuhiro Yamashita, and Yoshihide Nomura. 2019. A Solution for the Risk of Non-deterministic Transactions in Hyperledger Fabric. *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (2019), 253–261.

[1] https://github.com/ninori9/GraphGenerationFrontend

[2] https://drive.google.com/file/d/1sa1FcNxPHV1fEkmmpsZzPTk7i0NOd5GF

ACM Publishing License and Audio/Video Release

Title of the Work: Fabric-Visualizer: A Transaction Dependency Visualizer for Hyperledger Fabric
Submission ID:middleware23dpds-p82

Author/Presenter(s)· **Jeeta Ann Chacko·Technical University of Munich;Nino Richter·Technical University of Munich;Ruben Mayer·University of Bayreuth;Hans-Arno Jacobsen·University of Toronto**

Type of material:short paper

Publication and/or Conference Name: Middleware Demos, Posters and Doctoral Symposium'23:24th International Middleware Conference Demos, Posters and Doctoral Symposium Proceedings

1. Glossary

2. Grant of Rights

(a) Owner hereby grants to ACM an exclusive, worldwide, royalty-free, perpetual, irrevocable, transferable and sublicenseable license to publish, reproduce and distribute all or any part of the Work in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library, and to authorize third parties to do the same.

(b) In connection with software and "Artistic Images and "Auxiliary Materials, Owner grants ACM non-exclusive permission to publish, reproduce and distribute in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library.

(c) In connection with any "Minor Revision", that is, a derivative work containing less than twenty-five percent (25%) of new substantive material, Owner hereby grants to ACM all rights in the Minor Revision that Owner grants to ACM with respect to the Work, and all terms of this Agreement shall apply to the Minor Revision.
(d) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

☑ A. Grant of Rights. I grant the rights and agree to the terms described above.

□ B. Declaration for Government Work. I am an employee of the national government of my country/region and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are you a contractor of your National Government? ○ Yes ● No

Are any of the co-authors, employees or contractors of a National Government?
○ Yes ● No
Are all of the co-authors, employees or contractors of a National Government? ○ Yes
● No

3. Reserved Rights and Permitted Uses.

(a) All rights and permissions the author has not granted to ACM in Paragraph 2 are reserved to the Owner, including without limitation the ownership of the copyright of the Work and all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM in Paragraph 2(a), Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference. When creating your document, please make sure that you are only using TAPS accepted packages. (If you would like to use a package not on the list, please send suggestions to acmtexsupport@aptaracorp.com RE: TAPS LaTeX Package evaluation.)

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please put the following LaTeX commands in the preamble of your document - i.e., before \begin{document}:

```
\copyrightyear{2023}
\acmYear{2023}
\setcopyright{acmlicensed}\acmConference[Middleware Demos, Posters and Doctoral Symposium'23 ]{24th International Middleware Conference Demos, Posters and Doctoral Symposium}{December 11--15, 2023}{Bologna, Italy}
\acmBooktitle{24th International Middleware Conference Demos, Posters and Doctoral Symposium (Middleware Demos, Posters and Doctoral Symposium'23 ), December 11--15, 2023, Bologna, Italy}
\acmPrice{15.00}
\acmDOI{10.1145/3626564.3629098}
\acmISBN{979-8-4007-0429-1/23/12}
```

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

If you are using the ACM Interim Microsoft Word template, or still using or older versions of the ACM SIGCHI template, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

Middleware Demos, Posters and Doctoral Symposium'23 , December 11–15, 2023, Bologna, Italy

ACM.
ACM ISBN 979-8-4007-0429-1/23/12...$15.00
https://doi.org/10.1145/3626564.3629098

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF to your event contact for processing.

---

4. ACM Citation and Digital Object Identifier.

(a) In connection with any use by the Owner of the Definitive Version, Owner shall include the ACM citation and ACM Digital Object Identifier (DOI).
(b) In connection with any use by the Owner of the Submitted Version (if accepted) or the Accepted Version or a Minor Revision, Owner shall use best efforts to display the ACM citation, along with a statement substantially similar to the following:

> "© [Owner] [Year]. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in {Source Publication}, https://doi.org/10.1145/{number}."

5. Livestreaming and Distribution

You are giving a presentation at the annual conference. This section of the rights form gives you the opportunity to grant or deny ACM the ability to make this presentation more widely seen, through (a) livestreaming of the presentation during the conference and/or (b) distributing the presentation after the conference in the ACM Digital Library, the "Conference Presentations" USB, and media outlets such as Vimeo and YouTube. It also provides you the opportunity to grant or deny our use of the presentation in promotional and marketing efforts after the conference.

Not all conference presentations are livestreamed; you will be notified in advance of the possibility of your presentation being livestreamed.

The permissions granted and/or denied here apply to all presentations of this material at the conference, including (but not limited to) the primary presentation and any program-specific "fast forward" presentations.

ACM's policy on the use of third-party material applies to your presentation as well as the documentation of your work; if you are using others' material in your presentation, including audio, you must identify that material on the ACM rights form and in the presentation where it is used, and secure permission to use the material where necessary.

Livestreaming.
I grant permission to ACM to livestream my presentation during the conference (a "livestream" is a synchronous distribution of the presentation to the public, separate

from the presentation distributed to conference registrants).
● Yes
○ No

Post-Conference Distribution.
I grant permission to ACM to distribute the recording of my presentation after the conference as listed above.
● Yes
○ No

6. Auxiliary Material

Do you have any Auxiliary Materials? ○ Yes ● No

7. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

● We/I have not used third-party material.
○ We/I have used third-party materials and have necessary permissions.

8. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part IV and be sure to include a notice of copyright with each such image in the paper.
● We/I do not have any artistic images.
○ We/I have any artistic images.

---

9. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants.

10. Enforcement.

At ACM's expense, ACM shall have the right (but not the obligation) to defend and enforce the rights granted to ACM hereunder, including in connection with any instances of plagiarism brought to the attention of ACM. Owner shall notify ACM in writing as promptly as practicable upon becoming aware that any third party is infringing upon the rights granted to ACM, and shall reasonably cooperate with ACM in its defense or enforcement.

11. Governing Law

This Agreement shall be governed by, and construed in accordance with, the laws of the state of New York applicable to contracts entered into and to be fully performed therein.

Funding Agents

1. Bavarian Cooperative Research Program of the Free State of Bavaria award number(s):DIK-2002-0013//DIK0114/02

2. Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) award number(s):392214008

DATE: **10/26/2023** sent to chacko@in.tum.de at **08:10:41**