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

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

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.

ACM Reference Format:

Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2023. How To Optimize My Blockchain? A Multi-Level Recommendation Approach. In *Proceedings of ACM Conference (Conference'23)*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnnnnn

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].

Conference'23, June 2023, Seattle, WA, USA

© 2023 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

https://doi.org/10.1145/nnnnnnnnnnnn



Figure 1: Multi-level blockchain optimization

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 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-bycase 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

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 ACM 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.

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 systemlevel 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 Block-OptR 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 systemlevel 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:

- 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.



Figure 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.

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		

Figure 3: Transaction dependency conflict example

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 NPhard problem of generating conflict-free dependency graphs [67]. An increase in endorsement policy failures due to inconsistent world 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.



Figure 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.

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



Figure 5: BlockOptR workflow

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 userconfigurable 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 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

Table 1: Formalization of optimization recommendations

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

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



Figure 6: Optimization implementation on a live Fabric network

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.

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

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%	

Table 2: Control variables

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. *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)

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

Table 3: Experiments with the synthetic workload

Experiment Number	Control variable	Value	Optimizations recommended	
1	Endorsement	P1	Endorser restructuring	
	Policy		Activity reordering	
2	Endorsement Policy /	P2 / 6	Endorser restructuring	
	Endorser dist skew		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	
			Transaction rate control	
8	Key		Activity reordering	
	distribution skew	2	Smart contract partitioning	
			Block size adaptation	
9	Block count	50	Activity reordering	
			Transaction rate control	
10	Block count	300	Activity reordering	
			Transaction rate control	
11	Block count	1000	Activity reordering	
12	Send rate	50	Activity reordering	
13	Send rate	300	Activity reordering	
			Block size adaptation	
			Transaction rate control	
14	Send rate	1000	Activity reordering	
			Transaction rate control	
15	Transaction	70%	Activity reordering	
	distribution skew		Client resource boost	

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)

Thirdly, we created a smart contract and workload using a reallife 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

Table 4: Settings to implement optimization

Optimizations	Settings	
Recommended		
Activity reordering	Reorder workload generation	
Transaction rate control	Set send rate to 100 TPS	
Process model pruning		
Delta writes	Update smart contract	
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	

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 4.

6.1 Synthetic Workloads

Due to space restrictions, we present 15 workloads in Table 3. 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 3. 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.

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 Block-OptR 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.

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.

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



Figure 12: All recommended optimizations combined

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 3). 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 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.

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.



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 (query-Products 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



Figure 18: Synthetic workloads with FabricSharp

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%.

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.

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



Figure 19: Synthetic workloads with Fabric++

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.

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 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 Ethereumbased 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

- 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-complainsprociess-at-granada-city-council.pdf. (2020). [Online; accessed 07-July-2022].
- [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 (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. (2018). 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 cpe.5227.
- BlockOptR 2022. https://github.com/anonysubm/BlockOptR. (2022). [Online; accessed 13 -April-2022].
- [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 (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/. (2022). [Online; accessed 07-April-2022].
- [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 (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 (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. (2020). [Online; accessed 07-July-2022].
- [18] Corda 2022. https://docs.r3.com/en/platform/corda/4.7/open-source/keyconcepts-notaries.html. (2022). [Online; accessed 02-April-2022].
- [19] Data Requirements: Case ID 2020. https://fluxicon.com/book/read/dataext/#caseid. (2020). [Online; accessed 07-July-2022].
- [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 (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 (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. (2022). [Online; accessed 13-April-2022].
- [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. URI: 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 H\u00e4rder. 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/. (2020). [Online; accessed 07-April-2022].
- [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. (2018). 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. 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 (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 (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/lib/exe/fetch.php?media=2017:bpi2017_ winner_professional.pdf. (2017). [Online; accessed 04-April-2022].
- [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 (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 (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. (2022). [Online; accessed 07-April-2022].
- [71] Updating a channel configuration 2022. https://hyperledger-fabric.readthedocs. io/en/release-2.2/config_update.html. (2022). [Online; accessed 07-April-2022].
- [72] Upgrading a smart contract 2022. https://hyperledger-fabric.readthedocs.io/en/ release-2.2/deploy_chaincode.html#upgrading-a-smart-contract. (2022). [Online; accessed 18-July-2022].
- [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. (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/articlepdf/6/2/172/37064506/fjaa010.pdf