Workflow Management on BFT Blockchains

Joerg Evermann*[,a], Henry Kim[b]

[a] Memorial University of Newfoundland, St. John's, Canada
[b] York University, Toronto, Canada

Abstract. *Blockchains have been proposed as infrastructure technology for a wide variety of applications. They provide an immutable record of transactions, making them useful when business actors do not trust each other, and their distributed nature makes them suitable for inter-organizational applications. However, widely-used proof-of-work based blockchains are computationally inefficient and do not provide final consensus, although they scale well to large networks. In contrast, blockchains built around Byzantine Fault Tolerance (BFT) consensus algorithms are more efficient and provide immediate and final consensus, but do not scale well to large networks. We argue that this makes them well-suited for workflow management applications, which typically include no more than a few dozen participants. This paper is motivated by a use case in the resource extraction industry. We develop an architecture for a BFT blockchain based workflow management system (WfMS) and present a prototype implementation. We discuss its advantages and limitations with respect to proof-of-work based systems and provide an outlook to future research.*

Keywords. Byzantine fault tolerance • blockchain • workflow management • interorganizational workflow • distributed workflow

1 Introduction

Inter-enterprise business processes may include stakeholders in adversarial relationships, that nonetheless have to jointly complete process instances. Trust in the current state of a process instance and correct execution of activities by other stakeholders may be lacking. Blockchain technology can help in such situations by providing a trusted, distributed workflow execution infrastructure.

A blockchain cryptographically signs a series of blocks, containing transactions, so that it is difficult or impossible to alter earlier blocks in the chain. In a distributed blockchain, actors independently validate transactions, add them to the blockchain, and replicate the chain across different nodes. The independent and distributed nature of actors requires finding a consensus regarding the validity and order of transactions and blocks. In workflow execution, it is important that actors agree on the "state of work" as this determines the set of next valid activities in the process. Hence, it is natural to use blockchain transactions to describe workflow activities or workflow states.

Blockchain technology admits many different system designs, and WfMS can be implemented in many different ways on different kinds of blockchain infrastructure. In this paper we explore a novel design for blockchain-based workflow management. Our contributions are twofold:

First, in contrast to prior work, which has focused on proof-of-work blockchains, we examine the use of consensus protocols based on algorithms for Byzantine Fault Tolerance (BFT), recommended by Viriyasitavat and Hoonsopon (2018), and show how a BFT-based blockchain can be used as workflow management infrastructure. BFT based blockchains are computationally more

* Corresponding author.
E-mail. jevermann@mun.ca
Note: A draft version of this paper was published as a pre-print at http://arxiv.org/abs/1905.12652

efficient, provide fast and final ordering and consensus, and, depending on the use case, can offer better resilience to faults or attacks (Viriyasitavat and Hoonsopon 2018).

Second, in contrast to earlier work, we explore the architecture of a blockchain-based workflow management system (WfMS) without smart contracts. Not basing the WfMS on smart contracts allows the easy adaptation of existing workflow engines to the blockchain. Even without the use of smart contracts, the blockchain remains essential as it provides independent validation of workflow activities, distribution, replication, and tamper-proofing.

We describe a prototype WfMS system as a proof-of-concept implementation for an architecture that has not yet seen attention in the literature. We focus on the interface between the blockchain and the workflow engine and the architectural options available for the design of the system. While our prototype is an important demonstration of feasibility, our main contribution is in the identification and discussion of the different architectural choices, and highlighting the existence and feasibility of alternatives to smart contracts on proof-of-work blockchains.

2 Motivating Example and Use Case

Our motivating example is in the mining and resource extraction industry. Resource extraction involves different parties, including the proponent (the mining or extraction company), national, regional and municipal governments, various technical experts and consultants, and, in many countries, indigenous peoples or nations (Sengupta and Kim 2020). Indigenous peoples must be consulted as part of the approval process (OECD 2017). In Canada, for example, the duty to consult rests with the federal government. However, it in turn relies on the extraction industry stakeholders to carry out the consultation process appropriately (Canadian Chamber of Commerce 2016; Gray 2015). The approval and consultation process is complex and frequently suffers from a lack of trust, due in part to prior broken treaty promises (Sengupta and Kim 2020). Various guidelines exist for carrying out this process (Canadian Chamber of Commerce 2016; Government of Canada 2015; Gray 2015; Nishnawbe Aski Nation 2007; OECD 2017) but the precise process and the involved actors are specific to each project. In general, the process involves the preparation and consultation, working with federal departments and agencies, responding to government inquiries, environmental assessment, regulatory decision making, and ongoing relationship and commitments to indigenous groups (Government of Canada 2015). Each of these in turn involve multiple activities. To provide but one example, the Nishnawbe Aski Nation in Ontario, Canada has defined its process for consultation in 10 major phases (Nishnawbe Aski Nation 2007).

The important characteristics of this use case are the following:

1. Lack of trust, primarily by indigenous nations
2. Requirement to verify adequate process execution by all stakeholders
3. A small number of known actors
4. Resource disparity (extraction companies control relatively large amounts of financial and other resources)

Characteristics 1 and 2 suggest the use of blockchain technology to address the lack of trust and the verifiability requirements. Characteristic 3 suggests the use of a private blockchain. Characteristic 4 suggests that a proof-of-work blockchain is not suitable, as the extraction company could use its resources to undermine the integrity of such a blockchain. Together, characteristics 3 and 4 suggest the use of Byzantine Fault Tolerance as an alternative ordering and validation mechanism: It is relatively harder even for a single well-endowed stakeholder to control $1/3$ of the blockchain nodes than it is to amass $> 50\%$ of hashing power (cf. Sec. 5).

3 Conceptual Modeling of Inter-organizational Workflows on Blockchains

BPMN is the current standard for process modelling. Inter-organizational processes can be described using choreographies and collaborations. Figure 1 shows a high-level choreography diagram of the consultation process described by the Nishnawbe Aski Nation (2007). BPMN uses message passing for participants to interact and coordinate separate, independent processes. Each choreography task in Fig. 1 is equivalent to a pair of message sending and receiving tasks (or events) in a collaboration. The choreography in Fig. 1 is presented as an example of a choreography and to provide an impression of our motivating use case; it is too abstract for implementation and the choreography tasks need to be further decomposed.

While message passing and choreographies are useful in conceptually modelling multi-party interactions of independent participants and separate, independent processes, the use case we have described (Sec. 2) allows the inter-organizational process to be simplified. As all stakeholders jointly define the process, we define a single process rather than multiple independent processes that are coordinated by messages. Hence, we can describe the process as a BPMN diagram with a single pool that represents the set of organizations (stakeholders). In this view, we define each lane to represent a participating organization (Fig. 2). In the blockchain context, we define each atomic lane to also correspond to a blockchain node on which activities are executed. If a participating organization provides multiple blockchain nodes, this should be modelled by nested child lanes.

We note that the process in Fig. 2 is also a high-level description, and much additional detail is required for an executable process.

4 Related Work

This section discusses prior work in the two research areas our work is based on: blockchain technology applied to workflow management, and BFT ordering applied to blockchains.

4.1 Workflow Management and Blockchains

Blockchain-based workflow management has only recently received research attention (Mendling et al. 2018). Two research challenges are the conceptual modelling of blockchain-based workflows and the use of blockchain infrastructure for workflow execution (Mendling et al. 2018).

Blockchains and Conceptual Modelling of Workflows

Blockchains can be used as infrastructure for the collaborative modelling of workflows. The system described by Härer (2018) allows distributed, versioned modelling of private and public workflows, consensus building on versions to be instantiated, and tracking of instance states on the blockchain. The blockchain provides integrity assurance for models.

Another approach to use blockchains for conceptual modelling are knowledge chains (Fill and Härer 2018). This approach provides a permission system for model modifications, a system for storing model elements in blockchain blocks, and can track knowledge provenance. It adapts generic proof-of-work blockchains to store enterprise and process model content together with domain-specific block validation rules.

The modelling of blockchain-based workflows may require blockchain-specific conceptual models. Because transactions on proof-of-work based blockchains are never irrevocably confirmed (though eventually considered to be sufficiently confirmed for acting on, cf. Sec. 5), the BlockME approach provides an extension to BPMN that allows modellers to deal with transaction state changes (Falazi et al. 2019a). BlockME also provides an interface between blockchains and the workflow engine, so that workflow tasks can receive messages about transaction status changes and create and submit new transactions. An extension (BlockME2) provides a measure for degree of confirmation in different blockchains and access to smart contracts (Falazi et al. 2019b).

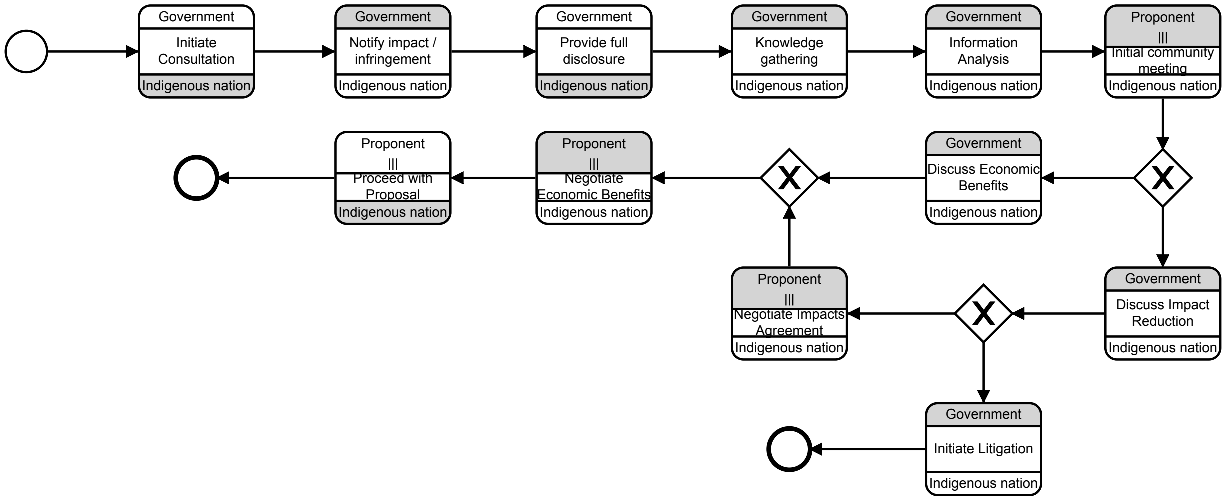


Figure 1: Choreography example for motivating use case (resource extraction consultation process)

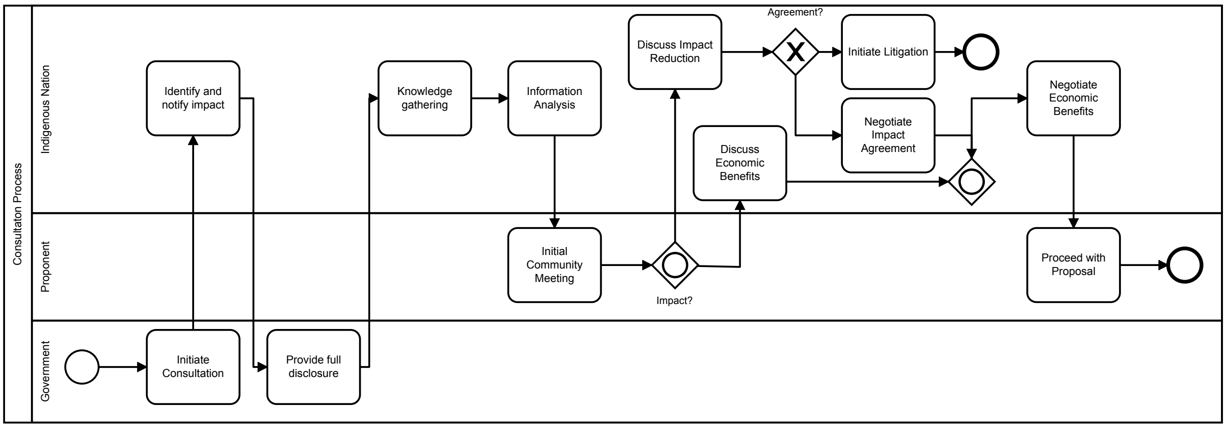


Figure 2: Collaboration example for motivating use case (resource extraction consultation process)

Blockchains and Execution of Workflows

Blockchains can be used as infrastructure for the execution of workflows. A number of workflow execution prototypes implementations have been presented, focusing on the use of "smart contracts". A smart contract is a software application that is recorded and executed on the blockchain. This application "listens" for relevant transactions sent to it and executes application logic upon receipt of a transaction. For example, the widely used Ethereum[1] blockchain has a Turing-complete virtual machine (VM) for smart contracts and compilers for different programming languages.

In a project driven by a financial institution, a workflow implementation using Ethereum and smart contracts supports digital document flow in the import/export trading domain (Fridgen et al. 2018, 2017). The project demonstrates lowered execution cost, and claims increased transparency and trust among trading partners.

Hukkinen et al. (2017) describe a blockchain-based workflow project in the real-estate domain, also using Ethereum and smart contracts. Hukkinen et al. (2017) claim that the lack of a central agency will make it difficult for regulators to enforce obligations and responsibilities of trading partners.

[1] https://ethereum.github.io/yellowpaper/paper.pdf

A blockchain-based workflow execution system by Weber et al. (2016a,b) uses Ethereum smart contracts either as choreography monitors, where the smart contract monitors execution status and validity of workflow messages against a process model, or as an active mediator, where the smart contract controls the process by sending and receiving messages[2] according to a process model. BPMN models are translated into smart contracts. Local nodes monitor the blockchain for relevant messages from the smart contract and create transactions for the smart contract. The cost for executing smart contract transactions and the execution latency are recognized as important considerations in the evaluation of the approach. A comparison between the public Ethereum blockchain and the Amazon Simple Workflow Service shows blockchain-based costs to be two orders of magnitude higher than a traditional infrastructure (Rimba et al. 2017, 2018). Hence, optimizing the space and computational requirements for smart contracts is important (García-Bañuelos et al. 2017). BPMN models are first translated to Petri Nets (Dijkman et al. 2008), for which minimizing algorithms are known. The minimized Petri nets are then compiled into smart contracts, achieving up to 25% reduction in execution costs for the required Ethereum blockchain transactions (Weber et al. 2016a,b), while also significantly improving throughput time. Building on lessons learned from Weber et al. (2016a,b), Caterpillar is an open-source blockchain-based workflow management system (López-Pintado et al. 2017). Developed using the Node.js JavaScript runtime it uses standard Ethereum tools, like the Solidity compiler *solc* and the Ethereum client *geth*, to provide a distributed execution environment for BPMN-based process models. The Caterpillar system has recently been extended to directly interpret BPMN models. In other words, it provides a workflow management engine developed as a set of Solidity smart contracts for Ethereum, rather than translating individual BPMN models to model-specific smart contract. Lorikeet (Ciccio et al. 2019) is similar to the original Caterpillar system, also based on BPMN models that are translated to smart contracts for the Ethereum blockchain.

Also working with Ethereum, Sturm et al. (2019) present a system focusing on resource management and extends smart contracts to implement a range of resource allocation patterns.

While most implementations use a flow-based workflow specification, declarative workflows can also be deployed on a blockchain infrastructure (Madsen et al. 2018). This approach is also implemented on the Ethereum platform using Solidity smart contracts.

The replicated nature of blockchains means that information is available to all participants. One approach to address this privacy issue in the context of workflow management is the use of access control lists and their enforcement in smart contracts (Pourheidari et al. 2018) or the use of encryption and key exchange patterns to limit visibility of information in smart contracts (Köpke et al. 2019).

Summary and Comparison

With respect to conceptual modeling, in contrast to (Falazi et al. 2019a,b), our work is not based on proof-of-work blockchains, but on BFT-based blockchains with immediate and final ordering and consensus. Hence, modelling of blockchain transaction status is not required. In contrast to (Härer 2018), our system does not include a collaborative modeling component; this is outside of our research scope.

With respect to workflow execution, while existing work varies in terms of architecture, features and capabilities, *all existing blockchain-based workflow execution systems are based on proof-of-work blockchains* and *all use smart contracts*. More specifically, *all are based on the Ethereum blockchain* and its smart contract virtual machine.

[2] An Ethereum transaction is a message to an externally-owned account or to an autonomous object (smart contract) that is signed by an externally-owned account using its public key identifier. Smart contracts are autonomous objects that can call other contracts by sending messages. As they do not possess a public/private key pair for signing messages, those messages are not considered transactions.

In contrast, our work is based neither on proof-of-work chains, nor on smart contracts.

4.2 BFT Ordering in Blockchains

Solving the transaction ordering and consensus problems not with computationally expensive proof-of-work approaches but with efficient and provably correct and live algorithms is an important motivator for many blockchain projects. The Hyperledger project of the Linux foundation is an umbrella for a number of BFT-based blockchain implementations. Hyperledger Burrow[3] is a blockchain that can execute Ethereum virtual machine code but is based on the Tendermint[4] BFT-based consensus algorithm. Hyperledger Iroha[5] is based on "YAC", a proprietary BFT-based consensus protocol, but does not provide smart contracts. Hyperledger Indy[6] is a blockchain implementation for decentralized identity management, based on redundant byzantine fault tolerance (RBFT) (Aublin et al. 2013). Hyperledger Fabric[7] is a generic blockchain implementation that provides smart contracts, called "chaincode", which can be written in Go or JavaScript using the Node.js runtime. However, while early implementations used the BFT-SMART ordering protocol (Sousa et al. 2018), recent versions have moved to the simpler, crash-fault tolerant (CFT) RAFT algorithm (Ongaro and Ousterhout 2014).

Examining different blockchain consensus mechanisms in terms of termination time and fault tolerance, Viriyasitavat and Hoonsopon (2018) recommend BFT-based consensus for workflow execution because "it guarantees safety, liveness, and some degree of fault tolerance" and proof-of-work is "impractical since the confirmation settlement is too long and unreliable". Additionally, proof-of-work is inefficient as it is essentially a competition to expend the most hashing power. This also creates sustainability problems: For example, Ethereum's energy cost per transaction is approx. 33 kWh for a total annual electricity consumption of over 7 TWh[8] .

5 Blockchains

A blockchain records transactions in contiguous blocks. A transaction can be any kind of content. Integrity is maintained by applying a hash function to the content of each block, which also contains the hash of the previous block in the chain. Hence, altering a block requires changing all following blocks. In a distributed blockchain, nodes are connected using a peer-to-peer network topology. New transactions may originate on any node and must be recorded in new blocks. Blocks are typically distributed to each node for independent validation and replicated storage. The key challenge is to achieve a consensus on the validity and order of transactions and blocks, despite nodes that are characterized by "byzantine faults" (Lamport et al. 1982): they may not respond correctly, may respond unpredictably, or may become altogether unresponsive.

5.1 Public and Permissioned Blockchains

Blockchains may be either public or permissioned ("consortium"). Public blockchains typically have no access control or identity management. Hence, no node can be assumed to be trustworthy. In contrast, a permissioned blockchain has access controls, node operators are generally known and invited to participate, and (some) node operators may be implicitly trusted. The distinction between public and permissioned is not binary, but a continuum (Viriyasitavat and Hoonsopon 2018).

Public chains are typically created to serve a large number (thousands or millions) of anonymous participants. Their advantages include anonymity, universal access, and high trustworthiness as a large number of nodes provide independent transaction validation. On the other hand, public chains require incentives for validation, often in the form of a cryptocurrency, increasing

[3] https://www.hyperledger.org/projects/hyperledger-burrow
[4] https://tendermint.com/
[5] https://www.hyperledger.org/projects/iroha
[6] https://www.hyperledger.org/projects/hyperledger-indy
[7] https://www.hyperledger.org/projects/fabric
[8] https://digiconomist.net/ethereum-energy-consumption, retrieved on 23 Jan 2020

transaction costs. Public chains also provide little flexibility to adapt to special use cases.

In contrast, permissioned chains are typically created for specific use cases with a small number (tens or hundreds) of known participants (Viriyasitavat and Hoonsopon 2018). Advantages of permissioned chains include low transaction costs, high flexibility to adapt to special use cases, identifiability of transaction originators, and access controls. Disadvantages may include relatively lower trustworthiness due to the smaller number of validating nodes, in particular for proof-of-work based blockchains.

Workflow management, and our motivating use case (Sec. 2), is typically the domain of a small number of institutional collaborators, rather than a large number of anonymous participants. As such, it is a good fit with permissioned blockchains.

5.2 Smart Contracts versus Application Code

Smart contracts allow code execution as part of transactions on the blockchain. Advantages include code integrity, as code is part of the blockchain, and a tight integration of application logic with transaction validation. Disadvantages may be limitations of the smart contract language instruction set and the need to re-develop existing application logic.

In contrast, implementing application logic off-chain means that existing applications do not need to be ported, and developers have access to familiar programming languages, code libraries and development tools. On the other hand, transaction validation must call back to the application logic.

Smart contracts ensure that all nodes provide the same validation results, whereas performing validation in off-chain logic places the onus on the developers to ensure identical results for all nodes. On the other hand, it allows developers to develop against a behavioural specification without specifying the exact algorithms or implementation to be used. For WfMS, this means that transparency is lost about the specific details of the workflow implementation, but what is gained is that different workflow systems can interoperate as long as all obey the same behavioural semantics of the workflow specification language; in other words, they agree on which actions are permissible in a particular workflow state.

Smart contracts have great potential in the context of workflow management, as witnessed by the the Caterpillar and Lorikeet approaches (Ciccio et al. 2019). However, neither Caterpillar in its original version, nor Lorikeet provide a BPMN based generic workflow engine as a smart contract. Both systems compile individual BPMN to specific smart contracts. *Given the extensive investment in WfMS by researchers and practitioners, we believe that investigating if and how standard WfMS can be implemented on top of blockchain infrastructure* without re-implementation *in smart contract languages is worthwhile.*

5.3 Proof-of-Work Consensus

Bitcoin popularized proof-of-work for consensus finding and securing the blockchain. New transactions are distributed to all nodes, validated, and added to a transaction pool. Each node can independently propose new blocks based on its latest block and distribute these to other nodes. Depending on network speeds and topology, each node may have a different set of blocks and transactions, and hence may propose different blocks, leading to *side branches*. Each node considers the longest branch as its current main branch and proposes new blocks based on it. Transactions in side branches are not considered valid. When a side branch becomes longer than the current main branch, the chain undergoes a *reorganization*. What was the side branch is validated and becomes the main branch. What was the main branch is considered invalid and becomes a side branch. Transactions no longer in the main branch are added back to the transaction pool. Hence, different nodes may consider different blocks and transactions as valid, but as proposed blocks are distributed across the network, nodes will eventually converge on a consensus regarding the valid blocks and transactions and their order in the main branch of the chain.

To limit the rate of new block proposals and to secure the blockchain against attacks, block proposers must solve a hard problem ("proof-of-work", "mining"). Typically, this is to require the block hash to be less than a certain value. A limited block rate allows nodes to achieve eventual consensus, and a hard problem prevents attackers from "overtaking" the creation of legitimate blocks with fraudulent one. Hence, a successful attack requires control of $> 50\%$ of the total hashing power of all nodes.

The probability for a transaction in the main branch to become invalid decreases with each block that is mined on top of it, but in principle it is always possible for a block to become invalid. Blockchain communities use rules of thumb for the number of additional blocks required to consider a transaction safe enough to act on. In addition to the lack of finality of consensus, this approach induces significant latency as applications must wait not only for one block but many to be created. Furthermore, applications must actively monitor the status of all transactions of interest, must react to chain reorganizations, and communicate these aspects to the user.

5.4 BFT-Based Consensus and State Machine Replication

In response to the drawbacks of the proof-of-work consensus, i. e. latencies, no finality of consensus, and required processing power, provably correct ordering algorithms based on distributed systems research have seen a resurgence in interest. Most of the ongoing research can be traced back to a practical method for achieving byzantine fault tolerance (PBFT) (Castro and Liskov 2002; Lamport et al. 1982). PBFT achieves consensus on the order of requests using a set of fully-connected ordering nodes. Tolerating up to f faulty nodes requires $3f + 1$ total nodes. However, resilience is not directly comparable to proof-of-work consensus that requires $> 50\%$ of hashing power for a successful attack. Especially in small networks, computing power may be easily concentrated in a single high-powered node, but gaining control over $1/3$ of nodes is difficult when there is lack of trust among participants, as in our motivating example (Sec. 2).

Protocol

PBFT is a three-stage protocol. Every ordering consensus is established by a specific set of nodes ("view"), with a leader or primary node. A client sends a request to all nodes. The leader proposes a sequence number for the request and broadcasts a *pre-prepare* message. Upon receipt of a *pre-prepare* message, a node broadcasts a corresponding *prepare* message if it has itself received the request, has not already received another pre-prepare message for the same sequence number, and is in the current view. This indicates the node is prepared to accept the proposed sequence number. Nodes then wait to receive $2f$ matching *prepare* messages, indicating that $2f+1$ nodes are prepared to accept the proposed sequence number for the request. When a node has received $2f$ identical *prepare* messages, it broadcasts a *commit* message to all nodes. Each node then waits to receive $2f$ identical *commit* messages, indicating that $2f + 1$ nodes have accepted the proposed sequence number for the request. Upon committing, the node executes the request and sends a *reply* message to the client. The client waits for $2f + 1$ identical replies, which indicates that a consensus has been reached on the sequence number of the request.

The leader is not a central authority and is changed by consensus, also using a three-stage protocol. When a node suspects the leader is faulty (fails to propose, or sequence numbers too high or low), it broadcasts a *viewchange* message with information about the current state and pending requests. When a node has received $2f$ matching *viewchange* messages, it responds with a *viewchange-acknowledgment* message to the proposed new leader, if its current state and pending requests matches those of the *viewchange* messages. After receiving $2f$ *viewchange-acknowledgment* messages, the new leader broadcasts a *newview* message with the current agreed state and set of pending requests (cf. Castro and Liskov (2002) for details).

Consensus request sequencing is closely related to state machine replication (SMR): Every node maintains a state that can be changed by client requests. When every node begins with the same state and executes requests in the same order, the state machine is replicated.

BFT SMART

BFT-SMART (Bessani et al. 2014) is a software library built around the PBFT protocol and adds dynamic view reconfiguration (nodes can join and leave views), and the MOD-SMART (Sousa and Bessani 2012) collaborative state transfer system.

Collaborative state transfer is useful when nodes create state checkpoints at different times ("sequential checkpointing"). Due to the lack of multiple identical checkpoints, a simple quorum protocol cannot be used. Instead, "collaborative state transfer" (Bessani et al. 2013) provides checkpoint and log information from multiple nodes in a way that allows a new node to verify its correctness.

BFT-SMART provides a simple programming interface. The client-side interface allows submission of requests. Applications implement a server-side interface, encapsulating the state machine, that receives ordered requests in consensus sequence from the library for execution. Replies are sent back to the requesting client. Operation requests are simple byte arrays and opaque to the library, the client- and server-side applications must serialize and deserialize these in a meaningful way. View reconfigurations (adding or removing a node, or changing the level of fault tolerance) are special types of requests but are treated as any other request for ordering and consensus purposes.

For state management, the server-side application implements methods to fetch and set state snapshots, also serialized as byte arrays. Requests are logged and the state is periodically checkpointed. When a node joins a view, it is sent the latest checkpointed state using collaborative state transfer, and any requests after that checkpoint are then replayed, allowing the server state to catch up to the consensus state.

BFT-SMART has been proven to be correct and live, i. e. it will provide the same sequence of operations to all nodes and will not deadlock (Bessani et al. 2014). In terms of throughput, a system with four nodes ($f = 1$) has been shown to support more than 15,000 ordering requests (1kB size) per second with latencies around 10 milliseconds on a local network. The performance decreases linearly as fault tolerance (and hence the number of nodes) increases: A system with 10 nodes ($f = 3$) has been shown to support more than 10,000 requests per second (Bessani et al. 2014). These synthetic benchmarks do not indicate realistic application performance: There were neither client-side nor server-side applications, requests were only ordered and then dismissed. We report performance figures for our WfMS prototype in Sec. 7.

Summary

BFT-based ordering avoids the latency, lack of finality and computational demands of proof-of-work consensus. On the other hand, its three-stage protocol imposes significant communication overhead and requires fully-connected nodes. Fault tolerance in BFT increases linearly with the number of nodes, but performance decreases due to additional communication. Vukolić (2015) presents a comparison of proof-of-work and BFT consensus, shown in Table 1. *The different strengths and weaknesses of the two consensus mechanisms suggest that BFT-based ordering is a good fit with small, permissioned blockchains in the workflow management context.*

6 Architecture and Design Choices

The main component of a WfMS is the workflow engine, which interprets the workflow model and enables work items for manual execution or execution by external applications (Hollingsworth 1995). Prior work (Sec. 4) has deployed the workflow engine on the blockchain itself. For example, by compiling a workflow model to a smart contract, the contract forms a workflow engine for a specific workflow model. Alternatively, blockchains can be treated as a trusted infrastructure

	Proof-of-work	**BFT ordering**
Node identity (typically)	open, anonymous	permissioned, identified
Consensus finality	no	yes
Scalability (ordering nodes)	excellent	limited
Scalability (clients)	excellent	excellent
Throughput	limited	excellent
Latency	high	low
Correctness proof	no	yes

Table 1: Comparison between proof-of-work and BFT-based blockchains, adapted from Vukolić (2015)

layer for generic off-chain workflow engines, using the blockchain only for storing and sharing the state of work and achieving consensus on that state. To our knowledge, there has been no such implementation using BFT-based, or any other, ordering mechanisms.

6.1 Services

Ordering, block management, and the workflow engine are the three main services in our system architecture. In contrast to proof-of-work based blockchains, our architecture requires no mining service, no transaction service to manage pending transactions, and no virtual machine to execute or validate smart contract operations.

Ordering Service

The ordering service in our prototype uses the BFT-SMART library (Bessani et al. 2014). It consists of a client adapter and a server. The client adapter receives new transactions from clients and submits them to the ordering layer. Once ordered, the ordering layer submits the transactions in consensus sequence to the ordering service server in order to add them to the blockchain. The ordering service server maintains as its state information a record of the latest block hash and block number, as well as a queue of pending transactions. When a sufficient number of transactions has been collected, the ordering service creates a new block and clears the transaction queue. The ordering service server returns the hash of its state as a result to the originating client adapter and client, allowing clients to detect lack of consensus and also compare consensus state to their local ordering service server state. Clients can also request the latest block hash.

Block Service

The block service stores the blockchain, may exchange blocks with other nodes, and verifies the integrity of the blockchain.

The block service uses a peer-to-peer network for block exchange with new and recovering nodes. This network is distinct from the network layer of BFT-SMART and is not fully connected. Block exchange is required only when a node begins operation and enters an ordering view. At that point, the ordering service state is first updated through the BFT-SMART state replication mechanisms. The block service then compares its latest block to the latest hash it requests from the ordering service. The latter is assumed to be authoritative as it reflects consensus. Verification of the blockchain then proceeds backwards from the block with the latest hash. Any missing blocks are requested from other peers and verified prior to adding them.

Workflow Engine

The workflow engine maintains information about workflow instances (cases) and workflow model definitions. It receives workflow transactions from new blocks that are added to the chain, updating the state of each process instance and creating work items accordingly. Through the worklist, it manages user interactions with work items and execution of external applications by work items.

In principle, a system architecture can encompass l ordering services, m block services, and n

workflow engines, possibly distributed on different network nodes. However, as the absence of trust among participating actors is a key motivation for the use of blockchains, we assume that every participant requires its own independent workflow engine, block service, and ordering service, i. e. $l = m = n$. This assumption significantly simplifies the architecture and its implementation. It allows for tighter coupling of the three components using local method calls instead of network requests with their associated latency and serialization requirements. New blocks can be created and stored locally on each node from ordered transactions instead of being propagated on a peer-to-peer network, and valid transactions can be immediately executed on each node by the local workflow engine. Fig. 3 shows the architecture of our system.

6.2 Basic Operation

The red arrows labeled with numbers in Fig. 3 indicate the steps of handling a workflow transaction in our system:

1. The user or an external application completes a work item in worklist.
2. The transaction is created and passed to the ordering service client adapter.
3. The transaction is submitted to the ordering service.
4. The transaction is passed in consensus order to the ordering service server of all nodes.
5. Every ordering service server validates the ordered transaction with its workflow engine.
6. A new block is created and passed to the local block service.
7. The block service notifies the workflow engine of the new block and its transactions.
8. The workflow engine updates the state of running cases and creates new work items for the local worklist.
9. Every ordering service server responds to the requesting ordering service client adapter with its last block hash and hash of its transaction pool.

The green arrows labeled with letters in Fig. 3 indicate the block exchange mechanism when a peer node is started.

A. The block service queries the ordering service server for latest hash and transaction number.

B. If the block service determines it is missing blocks, it broadcasts a block request to all other nodes.

C. The block services receive block requests.

D. The block services assemble blocks into response message.

E. The block service receives requested blocks and verifies the blockchain integrity.

In step A, note that the ordering service is started before the block service and receives the latest hash and transaction number through state exchange from other nodes. The block request in step B contains the lower and upper block numbers required by the node. In step B, the block service begins by querying one random peer. If it receives no response, it queries an increasingly larger number of peers for blocks. In step D, other nodes only respond if they can satisfy at least the upper block number, as verification of a blockchain proceeds backwards. In step E, if the blockchain contains the most recent block but is missing individual earlier blocks, the block service will successively request these blocks from the peer it has most recently received blocks from. If this fails, it will again broadcast a query for a specific block. As fragments of the blockchain and individual blocks are added, the block services successively verifies the chain integrity beginning with the latest block and the last hash received from the ordering service.

6.3 Block Creation

In a typical proof-of-work blockchain, new blocks are created by a mining service and then distributed among nodes. The same is possible in our approach. New blocks can be passed as replies from the ordering service servers back to the ordering service client that requested the add-transaction operation which triggered the block creation. That

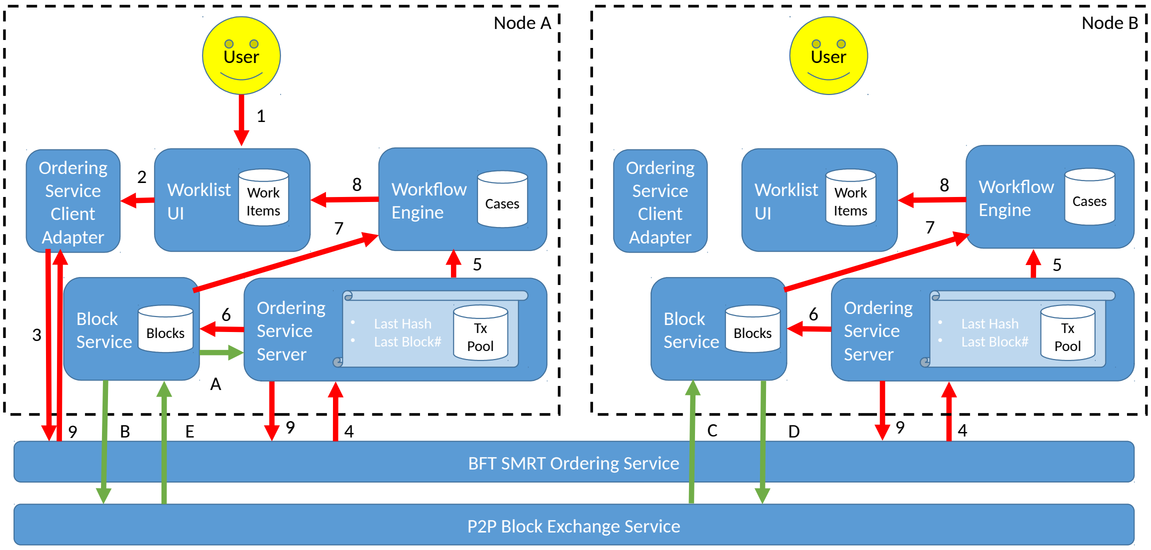


Figure 3: Architecture overview, transaction flow (red arrows, numbers), and block exchange mechanism at node startup (green arrows, letters)

node's block service then distributes the new block to other nodes using the peer-to-peer network. A more efficient alternative, made possible by the fact that every node contains both an ordering service and a block service, is to have every ordering service pass the new block directly to the local block service upon creation. As the order of requests is identical for all nodes, the blocks will be identical. This tighter coupling between ordering service and block service reduces the communication overhead for the peer-to-peer network and avoids latencies due to the block distribution. The peer-to-peer network is still required for block exchange with new or recovering nodes.

6.4 Block Size

In proof-of-work blockchains, blocks contain multiple transactions. The block size is a trade-off between desired transaction throughput, available hashing power, desired block creation rate, available network bandwidth, and tolerance for latency. A transaction may be "pending" for some time until it is included in a block and at a "safe" depth. In contrast, in BFT-based systems, there is no expensive mining and, in our architecture, no distribution of new blocks across a network. Hence, there is no reason to delay block creation and for blocks to contain multiple transactions: The blockchain becomes a chain of transactions.

Moving to a chain of transaction has another advantage. Proof-of-work systems order transactions between different blocks, but the order of transactions within a block is not defined: Transactions may be included in the same block as long as they are not mutually contradictory. Block miners ultimately impose an order, but this order is arbitrary. This means that as pending transactions are collected, they must be validated against the *entire* set of pending transactions to ensure they are not mutually conflicting. In a chain of transactions, a new transaction need to be validated only against the *immediately prior* one.

6.5 Ordering State

As the BFT-SMART library provides exchange of state information with new and recovering nodes, a decision must be made on what constitutes the state of each node. We considered two options. First, the state can comprise the entire blockchain. Second, the state can be reduced to comprise only the latest block hash. The first option means that the entire blockchain is part of the replicated state in BFT-SMART, removing the need for a peer-to-peer network with block exchange protocol. While easy to implement by serializing the blockchain into the BFT-SMART state snapshot, this option is feasible only for low-volume blockchains due to the complex and communication-intensive collaborative state transfer mechanism in BFT-SMART (i. e., a few megabyte). However, we anticipate larger volume for our motivating use case and typical workflow uses.

6.6 Workflow State or Workflow Operations

A transaction may represent *workflow operations* such as defining a new workflow model, launching a new case, executing an activity, aborting or cancelling a case, or removing a workflow model[9] . Upon activity execution by the user or an external application, the worklist handler submits an activity execution transaction that includes the activity name and case ID, as well as input and output data values. Once received by the workflow engines after ordering and inclusion in the blockchain, the workflow engines advance the state of the case as per the workflow specification semantics (e.g. workflow nets or BPMN).

Alternatively, a transaction may represent a *workflow instance state*, i. e. data values and enabled activities, without capturing the activity execution itself[10] . Upon activity execution by the user or an external application, the worklist handler advances the state of the case as per the workflow specification semantics (e.g. workflow nets or BPMN), and submits this new state as a transaction.

The first option requires the workflow engine to maintain its own state of the workflow (i. e. information about workflow models, running instances, data values and enabled activities). Constructing this state means reading the blockchain *forwards* from the genesis block and replaying transactions. State updates are done by executing transactions in new blocks. While reducing the amount of information stored on the blockchain, as only changed information is recorded, this option requires significant effort in managing the separate state and ensuring it is consistent with the blockchain record. However, this option is useful when an existing workflow engine that already maintains state is to be adapted to the blockchain infrastructure. In contrast, the second option makes the workflow state available by reading the blockchain *backwards* from the head to identify the latest state for each process instance. State updates are done simply by copying workflow states from blockchain transactions as new blocks are presented. Not maintaining separate state significantly simplifies the workflow engine design but leads to more information being stored on the chain. In contrast to public proof-of-work chains like Ethereum, where the transaction originator pays for storage and computation to incentivize mining, this is not necessary for private BFT-based blockchains.

6.7 Summary

In summary, the trust requirements for blockchains imply that every node includes an ordering service, a block service and a workflow engine. Using BFT-based ordering instead of proof-of-work simplifies issues such block creation and block size. The only significant design decision is whether to store workflow state or operations on the blockchain. This choice is primarily driven by the design of the workflow engines; existing engines are likely to work with operations, not complete workflow instance states. As we developed a custom workflow engine prototype to demonstrate our architecture, we chose to simplify the implementation and store workflow instance states.

7 Prototype Implementation

We implemented a prototype based on the architecture in Sec. 6 in Java. The source code is

[9] In this paper we focus on workflow execution; management and versioning of workflows models and updating of running cases are outside the scope of this work. Conceptually, they are easily added as another type of transaction to be handled by the workflow engine in the appropriate way.

[10] Here too, model management and versioning is conceptually easy to add as another transaction type to be handled.

available[11] , as well as a video demonstration[12] . Fig. 4 shows a screenshot of our prototype.

We implemented a permissioned peer-to-peer infrastructure with a pre-defined list of participating actors, each identified by a public/private key pair. To keep our prototype simple, actors are identified by their internet address rather than their public keys, so that we can omit an address resolution layer. The P2P layer is implemented using Java sockets and serialization. P2P and BFT-SMART ordering service messages are cryptographically signed and verified. Table 2 lists the message types on our peer-to-peer network.

Each P2P node has an outbound server that establishes connections to other peers, and an inbound server that accepts and verifies connection requests from peers. Each connection is served by a peer-connection thread, which in turn uses inbound and outbound queue handler threads to receive and send messages. Incoming messages are submitted to the inbound message handler which passes them to the appropriate service. Nodes can join and leave the peer-to-peer network at will. When a node joins, it tries to open connections to running peers. The first peer to be contacted will initiate a view change in the BFT-SMART ordering service to include the new peer on that layer as well.

Upon starting of a node, the BFT-SMART layer will first update state information from other nodes in the view. Next, the block service will identify missing blocks and request them from peers. Once the blockchain is complete and verified, the workflow engine reads the blockchain to get the latest state for each workflow instance.

Our system knows two transaction types. A *ModelUpdate* transaction installs a new workflow model definition. An *InstanceState* transaction contains a state of a workflow instance. It is submitted after a new case has been launched or a work item has been executed. Extensions to terminate cases and invalidate model definitions are readily possible. We have not included such extensions in order to focus on the relationship between workflow engine and blockchain infrastructure.

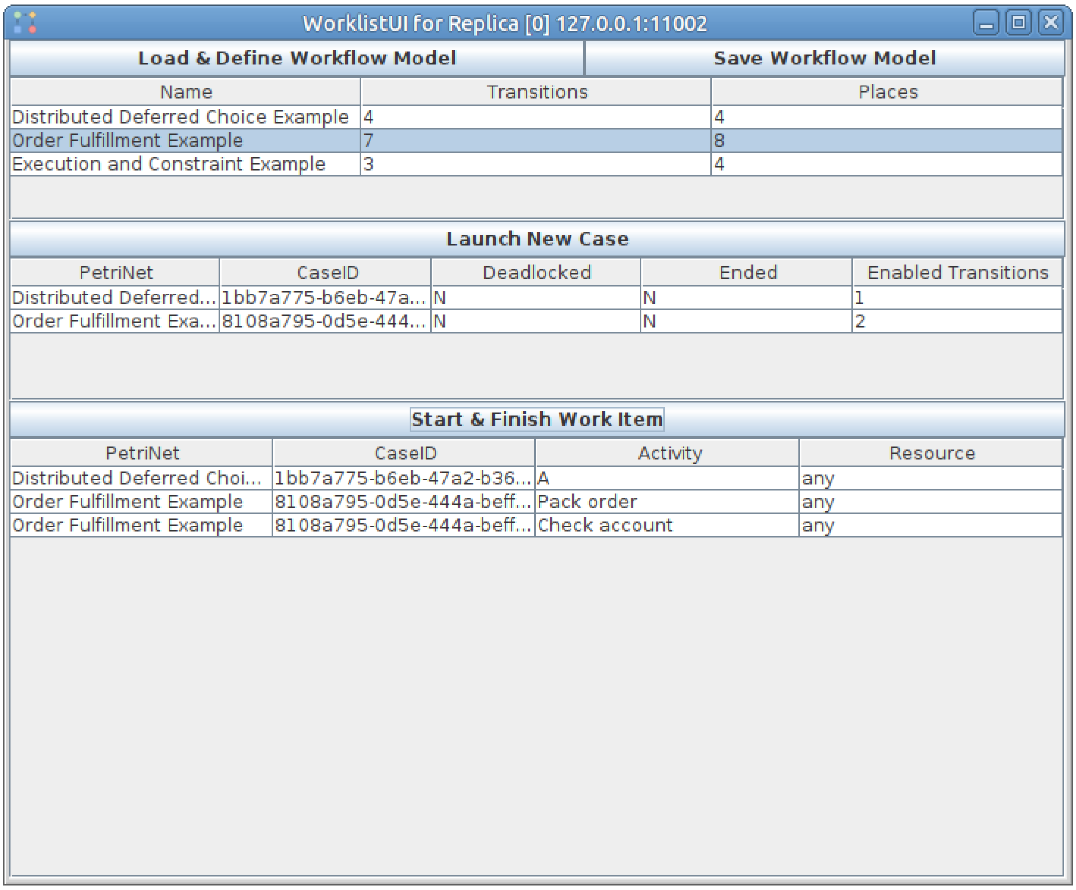
To keep our prototype simple, our workflow specifications are based on Petri nets (Aalst 1998). While Sec. 3 presented conceptual models using BPMN, much of BPMN, including multiple instances, messages, event-based gateways and exceptions, is conceptually translatable to Petri Nets (Dijkman et al. 2008). Each Petri net transition specifies a workflow activity. The workflow engine keeps track of the Petri net markings and case data, and can detect deadlocked and finished cases to remove them from the worklist.

Each activity is associated with a single node. Sec. 3 noted that BPMN lanes can be used to specify blockchain nodes. This partitioning of the process to different nodes does not form the entire resource perspective of the workflow but is used only to signal each node whether to act on a transaction. Each node can provide its own local resource management by defining roles or other organizational concepts and performing further work item allocation within each node. Our model specifications provide options for the process designer to specify this information.

External method calls are specified as calls to static Java methods, and are performed synchronously by the workflow engine upon work item enablement.

The data perspective is implemented as a key–value store. We currently admit only simple Java types as we implement a GUI for these; an extension to arbitrary types is readily possible. Each workflow instance has a set of data variables. When a transition is enabled, a work item is created for it and its input values are filled from the values of the workflow instance. The work item is then added to the local worklist or externally executed. After a work item is completed (manually or through execution of an external application), output values are written back to the workflow instance which is then submitted as an *InstanceState* transaction to the ordering service. The workflow designer can specify constraints on data values using arbitrary Java expressions.

[11] https://joerg.evermann.ca/software.html
[12] https://joerg.evermann.ca/BlockchainDemo.html


WorklistUI for Replica [0] 127.0.0.1:11002
Load & Define Workflow Model
Save Workflow Model
Launch New Case
Start & Finish Work Item

Name	Transitions	Places
Distributed Deferred Choice Example	4	4
Order Fulfillment Example	7	8
Execution and Constraint Example	3	4

PetriNet	CaseID	Deadlocked	Ended	Enabled Transitions
Distributed Deferred...	1bb7a775-b6eb-47a...	N	N	1
Order Fulfillment Exa...	8108a795-0d5e-444...	N	N	2

PetriNet	CaseID	Activity	Resource
Distributed Deferred Choi...	1bb7a775-b6eb-47a2-b36...	A	any
Order Fulfillment Example	8108a795-0d5e-444a-beff...	Pack order	any
Order Fulfillment Example	8108a795-0d5e-444a-beff...	Check account	any

Figure 4: Screenshot of prototype

BlockRequest	Requests a block with a specific hash from one or more peers
BlockSend	Sends a block to one or more peers
BlockChainRequest	Requests multiple blocks within a hash range from one or more peers
BlockChainSend	Sends multiple blocks to one or more peers

Table 2: Message types

Constraints are checked during validation of an *InstanceState* transaction.

The ordering service, workflow engine and the block service have a simple interface (Table 3). The ordering and block services can call on the workflow engine to validate transactions against the current workflow state. Validation checks that a workflow instance's new Petri net marking is reachable from the current Petri net marking of the workflow instance. Validation also checks the data constraints. The block service receives new blocks from the ordering service and passes them to the workflow engine. In the other direction, the workflow engine or the worklist UI can submit new transactions to the ordering service after a work item has been completed. Finally, the block service can request the latest hash from the ordering service on joining the network or recovering from a fault.

From the user's perspective, our system is little different from a traditional WfMS. In proof-of-work systems, the user or the workflow must be

→	validateTransaction(tx)	Ordering service asks workflow engine to validate a transaction (cf. arrow 5 in Fig. 3)
→	receiveBlock(block)	Block service receives a new block, stores it, and passes it to the workflow engine (cf. arrow 6 in Fig. 3)
←	addTransaction(tx)	Workflow engine or worklist UI submits a new transaction to the ordering service (cf. arrow 2 in Fig. 3)
←	getLatestHash()	Block service requests the latest hash from the ordering service (cf. arrow A in Fig. 3)

Table 3: Interfaces between ordering service, block service and workflow engine (directions from the perspective of the ordering service)

aware of and react to possible transaction invalidation, blockchain reorganization, eventual (delayed) consensus and transactions pending their required ”assume safe” mining depth, as witnessed by the BPMN extensions developed by Falazi et al. (2019a,b). In contrast, because of immediate and final consensus, our prototype behaves similar to traditional prototypes, with no pending transactions or latency for block mining. The execution status of workflow activities cannot change and need not be monitored or reported to the user. The user experiences a fast and responsive system, even in this prototype state.

8 Evaluation

This section evaluates our architecture and prototype on two dimensions. The first subsection examines performance, especially transaction throughput, while the second subsection evaluates the strength of the correctness guarantees made by our architecture and implementation.

8.1 Performance

While BFT-SMART has been subjected to synthetic benchmarks (Sec. 5.4), those indicate best-case performance and do not include application performance. In tests of our prototype workflow system, we achieved a throughput rate on a laptop computer with an Intel i7 4702HQ CPU (quad-core) and 16 GB RAM of approx. 85 workflow transactions per second. These transactions included launching cases (Petri nets with 7 transitions and 8 places) and performing workflow activities for each case to completion. We executed four WfMS nodes on the same machine, i. e. tolerating one faulty node. We have not yet profiled our prototype for code optimization. Improved performance could also be achieved by using separate machines on a local network or using the batch processing mechanism of BFT-SMART. However, our results compare well to those reported for a proof-of-work approach by Rimba et al. (2018), who report an average of 5.7 workflow transactions per second on a private Ethereum blockchain using an AWS m3.xlarge instance with more powerful CPUs.

While throughput (transactions per seconds) is a key performance metric for many blockchains and consensus algorithms, we believe it is less important in the workflow management context. Especially workflows with mostly manual activities and use cases with few participating actors (dozens), such as in our motivating example (Sec. 2), are unlikely to require a high sustained throughput. Even the largest organizations are unlikely to require sustained throughput of thousands of workflow transaction ordering operations per second.

8.2 Correctness Guarantees

Smart contract based systems like Caterpillar that deploy the workflow engine (model-specific or interpreted) on the blockchain enforce workflow consensus for every node as the workflow state is represented by the smart contract state. Submitting a transaction for an illegal workflow activity by a faulty or malicious node will cause the smart contract to retain the legal state and dismiss the workflow activity. Submitting a transaction with an invalid smart contract state will cause the transaction to be ignored by miners.

In contrast, our approach guarantees that the majority of nodes (and workflow engines) will arrive at a consensus about the current workflow state, assuming that the majority of nodes is not faulty or malicious (BFT approaches can tolerate up to 1/3 malicious nodes). When a node submits a transaction for an illegal workflow activity, the non-malicious majority nodes will each, individually and separately, reject the workflow activity. Hence, the majority consensus response is a "no change" response (specifically, the consensus ordering nodes return the hash of the last block instead of a new one), signaling that the workflow state has not changed. This indicates to the requester that it is either faulty or malicious. The requester can then, if faulty, request the correct BFT ordering state and rebuild its blockchain by transferring blocks from consensus nodes. A limitation of this approach is that faulty nodes can only detect their own fault once they submit a transaction; i. e. they cannot detect their own faults while they are only receiving transactions in new blocks.

When a node needs to catch up with the blockchain, the BFT-SMART state replication ensures that it receives the consensus last hash as state from the running ordering nodes. With this, it is able to detect incoming bad blocks as they are transferred. In general, assuming that there is a valid consensus on the ordering state (i. e. the last block hash), a node can always verify its blockchain and, if required, rebuild it by requesting blocks from other nodes.

They key difference between proof-of-work approaches and ours is that blocks are not created on a single node but on every node separately and concurrently. That is, the challenge is not to identify and reject bad (malicious) blocks as they are transferred, but only to ensure consensus ordering. The assumption of a majority of non-faulty nodes then ensures a majority of nodes with the correct workflow state.

9 Limitations

Throughput and Scalability

While our approach has lower latency and higher throughput, unlike proof-of-work chains it does not scale to a very large number of nodes; it is limited to an order of tens to (low) hundreds of nodes. Given these characteristics, architectures such as ours are suitable for permissioned blockchain applications using a small group of participants, as in our motivating use case (on the order of tens to hundreds). The low latency and high throughput also make them suitable for fast-moving processes, where activities are of short duration or must follow each other quickly. For example, our transaction throughput time is well below one second, whereas many proof-of-work blockchains operate at latencies on the order of minutes. However, our approach is not suitable for large or public blockchains.

Resilience

An often discussed type of attack on a proof-of-work based blockchain requires a malicious actor to control the majority ($> 50\%$) of the hashing power of all nodes. In contrast, attacking a BFT-based system requires control of more than 1/3 of all nodes. Under the assumption of equal hashing power in all nodes, the proof-of-work based blockchain appears more resilient to attacks. However, in many use cases, this assumption is unlikely to hold. Small networks and networks where a few actors control significant resources are particularly prone to an imbalance in hashing power. Our motivating use case (Sec. 2) is an example of such a situation where it is easy for a single actor (the mining company) to successfully attack a proof-of-work blockchain. In contrast, attacking a BFT-based system cannot be done by concentrating computational power but requires control of more than 1/3 of all nodes. This is difficult to achieve in the absence of trust among actors and in private or permissioned blockchains where new nodes cannot be introduced arbitrarily (see also Table 1 for a comparison). As a result, resilience to attacks and faults cannot be easily compared between proof-of-work and BFT-based blockchains; it is context and application dependent.

Workflow, Trust, and Fault Tolerance Requirements

In our approach, the number of nodes must strike a balance between the requirements of the workflow, the level of fault tolerance, and the performance of the system. The number of ordering nodes is determined by the desired level of fault tolerance, whereas the number of workflow nodes is determined based on the use case and the number of participating actors. A use case requiring more ordering than workflow nodes (e. g. because some actors share a workflow engine but do not wish to relinquish control over the trusted blockchain infrastructure) can be accommodated by nodes that are not assigned any workflow activities. On the other hand, when a use case requires more workflow nodes than ordering nodes (e. g. because groups of actors trust each other), the excess ordering nodes decrease performance due to the BFT protocol communication overhead. This drawback can only be addressed by relaxing the trust requirements, i. e. groups of actors must partially trust each other, so that the $1:1$ correspondence between ordering service, block service, and workflow engine can be relaxed.

Workflow Features and Language

Our implementation is not meant to be a fully-featured WfMS: It provides basic functionality to demonstrate in principle the feasibility of using BFT-based blockchains for workflow execution, and to study the interface and interplay of a workflow engine with blockchain infrastructure components. The WfMS features themselves are not the focus of this research. Hence, there are many limitations in our prototype workflow engine. While most BPMN modelling elements can be translated to the Petri Net workflow specifications used by our workflow engine, our workflow engine cannot receive messages from the external environment or handle timed events, both important BPMN modelling features. In particular, timed events may be a challenge for a distributed blockchain infrastructure that does not offer synchronized clocks; time synchronization on the blockchain is an active research field in itself (e. g. Fan et al. 2018). As noted above, model management is not currently performed by our prototype implementation. While versioning and distribution of workflow models itself is readily added, updating running cases is a complex issue in itself (e. g. Dias et al. 2003; Joeris and Herzog 1998; Kradolfer and Geppert 1999).

Public and Private Workflows

BPMN choreographies describe interactions of independent actors and define the public coordination points between otherwise private workflows. In contrast, in our motivating use case the actors jointly define a public process for consultation on resource extraction. The idea that the blockchain-based workflow system needs to capture only the public interactions of a larger public/private set of processes has not been discussed in the blockchain workflow literature or identified as a research challenge (Mendling et al. 2018). Consequently, work on systems like Caterpillar or Lorikeet, as well as this work, focuses on the execution of workflows on the blockchain, and assumes public workflow. The integration of blockchain-based public workflows with off-blockchain private workflows remains as a challenge for future research, both at the technology and at the conceptual modelling level.

10 Conclusions

Previous work on blockchain-based WfMS has focused on smart contracts and proof-of-work based blockchains. In particular, all prior work uses the Ethereum blockchain. However, proof-of-work-based systems have significant drawbacks in terms of processing power requirements, latency, and the lack of final consensus. In this work, we have shown that a BFT-derived ordering and consensus method is a suitable WfMS infrastructure. Even without the use of smart contracts, the use of a blockchain remains essential, as it provides independent validation of workflow actions, distribution, replication, and tamper-proofing to workflow management systems.

While there are limitations to the BFT-based approach (Sec. 9), our approach has significant advantages over proof-of-work based approaches:

- Our system is cheaper to operate than public proof-of-work blockchains that incentivize block mining through cryptocurrencies as there is no expensive mining that needs to be paid for, either as real electricity cost or as cryptocurrency exchange. While proof-of-work based blockchains may be deployed privately, they are then open to increased risk of attack (Sec. 9).
- Our system provides immediate and final consensus. This means that from both the workflow modeller's perspective and the user's perspective, the system looks and behaves like a traditional workflow engine. Neither the workflow designer nor the user need to deal with issues of transaction status or eventual transaction invalidation.
- Our system provides a greater throughput that proof-of-work based approaches.
- Not relying on smart contracts enables porting of existing feature-complete workflow engines, such as the open-source YAWL[13] (Hofstede et al. 2009) or Bonita[14] systems, to blockchain infrastructure. This allows a richer workflow language and leverages existing implementations. We have identified this aspect as our next research challenge.

To conclude, this paper has presented a prototype implementation for an architecture that has not yet seen any attention in the blockchain-based workflow literature. We have implemented a BFT-based system as recommended by Viriyasitavat and Hoonsopon (2018) and shown that this infrastructure is suitable for WfMS. We have shown how workflow engines can be fit onto a blockchain infrastructure without implementing them as smart contracts, paving the path to future work of adapting existing workflow engines to use blockchains as infrastructure for communication, persistence, replication, and trust building.

[13] http://www.yawlfoundation.org
[14] https://www.bonitasoft.com

References

van der Aalst W. M. P. (1998) The Application of Petri Nets to Workflow Management. In: Journal of Circuits, Systems, and Computers 8(1), pp. 21–66 https://doi.org/10.1142/S0218126698000043

Aublin P.-L., Mokhtar S. B., Quéma V. (2013) RBFT: Redundant Byzantine Fault Tolerance. In: IEEE 33rd International Conference on Distributed Computing Systems, ICDCS 2013, 8-11 July, 2013, Philadelphia, Pennsylvania, USA. IEEE Computer Society, pp. 297–306 https://doi.org/10.1109/ICDCS.2013.53

Bessani A. N., Santos M., Felix J., Neves N. F., Correia M. (2013) On the Efficiency of Durable State Machine Replication. In: Birrell A., Sirer E. G. (eds.) 2013 USENIX Annual Technical Conference, San Jose, CA, USA, June 26-28, 2013. USENIX Association, pp. 169–180 https://www.usenix.org/conference/atc13/technical-sessions/presentation/bessani

Bessani A. N., Sousa J., Alchieri E. A. P. (2014) State Machine Replication for the Masses with BFT-SMART. In: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014. IEEE Computer Society, pp. 355–362 https://doi.org/10.1109/DSN.2014.43

Canadian Chamber of Commerce (2016) Seizing six opportunities for more clarity in the duty to consult and accommodate process.

Castro M., Liskov B. (2002) Practical byzantine fault tolerance and proactive recovery. In: ACM Trans. Comput. Syst. 20(4), pp. 398–461 https://doi.org/10.1145/571637.571640

Ciccio C. D., Cecconi A., Dumas M., García-Bañuelos L., López-Pintado O., Lu Q., Mendling J., Ponomarev A., Tran A. B., Weber I. (2019) Blockchain Support for Collaborative Business Processes. In: Informatik Spektrum 42(3), pp. 182–190 https://doi.org/10.1007/s00287-019-01178-x

Dias P., Vieira P., Rito-Silva A. (2003) Dynamic evolution in workflow management systems. In: Proceedings of the 14th International Workshop on Database and Expert Systems Applications. IEEE, pp. 254–260

Dijkman R. M., Dumas M., Ouyang C. (2008) Semantics and analysis of business process models in BPMN. In: Information & Software Technology 50(12), pp. 1281–1294 https://doi.org/10.1016/j.infsof.2008.02.006

Falazi G., Hahn M., Breitenbücher U., Leymann F. (2019a) Modeling and execution of blockchain-aware business processes. In: SICS Software-Intensive Cyber-Physical Systems 34(2-3), pp. 105–116

Falazi G., Hahn M., Breitenbücher U., Leymann F., Yussupov V. (2019b) Process-Based Composition of Permissioned and Permissionless Blockchain Smart Contracts. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). IEEE, pp. 77–87

Fan K., Wang S., Ren Y., Yang K., Yan Z., Li H., Yang Y. (2018) Blockchain-based secure time protection scheme in IoT. In: IEEE Internet of Things Journal 6(3), pp. 4671–4679

Fill H.-G., Härer F. (2018) Knowledge blockchains: Applying blockchain technologies to enterprise modeling. In: Proceedings of the 51st Hawaii International Conference on System Sciences

Fridgen G., Radszuwill S., Urbach N., Utz L. (2018) Cross-Organizational Workflow Management Using Blockchain Technology - Towards Applicability, Auditability, and Automation. In: 51st Hawaii International Conference on System Sciences HICSS. AIS Electronic Library http://aisel.aisnet.org/hicss-51/in/blockchain/6

Fridgen G., Sablowsky B., Urbach N. (2017) Implementation of a Blockchain Workflow Management Prototype. In: ERCIM News 2017(110) https://ercim-news.ercim.eu/en110/special/implementation-of-a-blockchain-workflow-management-prototype

García-Bañuelos L., Ponomarev A., Dumas M., Weber I. (2017) Optimized Execution of Business Processes on Blockchain. In: Carmona J., Engels G., Kumar A. (eds.) Business Process Management - 15th International Conference, BPM, Proceedings. Lecture Notes in Computer Science Vol. 10445. Springer, pp. 130–146 https://doi.org/10.1007/978-3-319-65000-5%5C_8

Government of Canada (2015) Consultation and Accommodation Advice for Proponents.

Gray B. (2015) Building Relationships and Advancing Reconciliation through Meaningful Consultation.

Härer F. (2018) Decentralized Business Process Modeling and Instance Tracking Secured by a Blockchain. In: Bednar P. M., Frank U., Kautz K. (eds.) 26th European Conference on Information Systems ECIS. AIS Electronic Library, p. 55 https://aisel.aisnet.org/ecis2018%5C_rp/55

ter Hofstede A. H., van der Aalst W. M., Adams M., Russell N. (2009) Modern Business Process Automation: YAWL and its support environment. Springer Science & Business Media

Hollingsworth D. (1995) The workflow reference model.. Workflow Management Coalition

Hukkinen T., Mattila J., Seppälä T., et al. (2017) Distributed Workflow Management with Smart Contracts.. The Research Institute of the Finnish Economy

Joeris G., Herzog O. (1998) Managing evolving workflow specifications. In: Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (Cat. No. 98EX122). IEEE, pp. 310–319

Köpke J., Franceschetti M., Eder J. (2019) Balancing Privity and Enforceability of BPM-Based Smart Contracts on Blockchains. In: BPM (Blockchain and CEE Forum). Lecture Notes in Business Information Processing Vol. 361. Springer, pp. 87–102

Kradolfer M., Geppert A. (1999) Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In: Proceedings Fourth IFCIS International Conference on Cooperative Information Systems. CoopIS 99 (Cat. No. PR00384). IEEE, pp. 104–114

Lamport L., Shostak R., Pease M. (1982) The Byzantine Generals Problem. In: ACM Transactions on Programming Languages and Systems 4(3), pp. 382–401

López-Pintado O., García-Bañuelos L., Dumas M., Weber I. (2017) Caterpillar: A Blockchain-Based Business Process Management System. In: Clarisó R., Leopold H., Mendling J., van der Aalst W. M. P., Kumar A., Pentland B. T., Weske M. (eds.) Proceedings of the BPM Demo Track co-located with 15th International Conference on Business Process Modeling. CEUR Workshop Proceedings Vol. 1920 http://ceur-ws.org/Vol-1920/BPM%5C_2017%5C_paper%5C_199.pdf

Madsen M. F., Gaub M., Høgnason T., Kirkbro M. E., Slaats T., Debois S. (2018) Collaboration among adversaries: distributed workflow execution on a blockchain. In: 2018 Symposium on Foundations and Applications of Blockchain

Mendling J., Weber I., van der Aalst W. M. P., vom Brocke J., Cabanillas C., et al. (2018) Blockchains for Business Process Management - Challenges and Opportunities. In: ACM Trans. Management Inf. Syst. 9(1), 4:1–4:16 https://doi.org/10.1145/3183367

Nishnawbe Aski Nation (2007) A Handbook on Consultation in Natural Resource Development..

OECD (2017) OECD Due Diligence Guidance for Meaningful Stakeholder Engagement in the Extractive Sector. https://dx.doi.org/10.1787/9789264252462-en. OECD Publishing, Paris

Ongaro D., Ousterhout J. K. (2014) In Search of an Understandable Consensus Algorithm. In: Gibson G., Zeldovich N. (eds.) 2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.. USENIX Association, pp. 305–319 https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

Pourheidari V., Rouhani S., Deters R. (2018) A Case Study of Execution of Untrusted Business Process on Permissioned Blockchain. In: IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData 2018, Halifax, NS, Canada, July 30 - August 3, 2018. IEEE, pp. 1588–1594 https://doi.org/10.1109/Cybermatics%5C_2018.2018.00266

Rimba P., Tran A. B., Weber I., Staples M., Ponomarev A., Xu X. (2017) Comparing Blockchain and Cloud Services for Business Process Execution. In: 2017 IEEE International Conference on Software Architecture, ICSA. IEEE Computer Society, pp. 257–260 https://doi.org/10.1109/ICSA.2017.44

Rimba P., Tran A. B., Weber I., Staples M., Ponomarev A., Xu X. (2018) Quantifying the Cost of Distrust: Comparing Blockchain and Cloud Services for Business Process Execution. In: Information Systems Frontiers, pp. 1–19

Sengupta U., Kim H. M. (2020) Business Process Transformation in Natural Resources Development using Blockchain: Indigenous Entrepreneurship, Trustless Technology, and Rebuilding Trust. In: Treibelmeier H., Silaber C., T. C. (eds.) Blockchain and Distributed Ledger Technology Use Cases: Applications and Lessons Learned. in press. Springer

Sousa J., Bessani A. N. (2012) From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation. In: Constantinescu C., Correia M. P. (eds.) 2012 Ninth European Dependable Computing Conference, Sibiu, Romania, May 8-11, 2012. IEEE Computer Society, pp. 37–48 https://doi.org/10.1109/EDCC.2012.32

Sousa J., Bessani A., Vukolic M. (2018) A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. In: 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018. IEEE Computer Society, pp. 51–58 https://doi.org/10.1109/DSN.2018.00018

Sturm C., Scalanczi J., Schönig S., Jablonski S. (2019) A Blockchain-based and resource-aware process execution engine. In: Future Generation Computer Systems 100, pp. 19–34 http://www.sciencedirect.com/science/article/pii/S0167739X18327158

Viriyasitavat W., Hoonsopon D. (2018) Blockchain characteristics and consensus in modern business processes. In: Journal of Industrial Information Integration Preprint

Vukolić M. (2015) The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In: International workshop on open problems in network security. Springer, pp. 112–125

Weber I., Xu X., Riveret R., Governatori G., Ponomarev A., Mendling J. (2016a) Using blockchain to enable untrusted business process monitoring and execution.. Technical Report UNSW-CSE-TR-201609, University of New South Wales

Weber I., Xu X., Riveret R., Governatori G., Ponomarev A., Mendling J. (2016b) Untrusted Business Process Monitoring and Execution Using Blockchain. In: Rosa M. L., Loos P., Pastor O. (eds.) Business Process Management - 14th International Conference, BPM, Proceedings. Lecture Notes in Computer Science Vol. 9850. Springer, pp. 329–347 https://doi.org/10.1007/978-3-319-45348-4%5C_19