StreamChain: Building a Low-Latency Permissioned Blockchain For Enterprise Use-Cases

Lucas Kuhring*
Otto GmbH & Co KG, Germany
lucas.kuhring@otto.de

Zsolt István*
TU Darmstadt, Germany
zsolt.istvan@cs.tu-darmstadt.de

Alessandro Sorniotti
IBM Research, Switzerland
aso@zurich.ibm.com

Marko Vukolić
Protocol Labs
marko@protocol.ai

***Abstract*—Permissioned blockchains are a promising technology for secure decentralized data management in business-to-business use-cases. In contrast to Bitcoin and similar public blockchains, which rely on Proof-of-Work for consensus and are deployed on thousands of geo-distributed nodes, business-to-business use-cases (such as supply chain management and banking) require significantly fewer nodes, cheaper consensus, and are often deployed in controlled environments with fast networking and low latency. However, permissioned blockchains often follow the architectural thinking behind their WAN-oriented public relatives, which results in end-to-end latency several orders of magnitude higher than necessary.**

In this work, we propose a fundamental shift in permissioned blockchain design, eliminating blocks in favor of processing transactions in a *streaming* fashion. This results in a drastically lower latency without reducing throughput or forfeiting reliability and security guarantees. To demonstrate the wide applicability of our design, we prototype StreamChain based on the Hyperledger Fabric, and show that it delivers latency two orders of magnitude lower than Fabric, while sustaining similar throughput. This performance makes StreamChain a potential alternative to traditional databases and, thanks to its streaming paradigm, enables further research, for instance, around reducing latency through relying on modern programmable hardware accelerators.

I. Introduction

Blockchains (distributed ledgers) offer strong data integrity and reliability properties in untrusted environments that make them worth considering for distributed data management use cases beyond crypto-currencies. Their adoption, in particular that of *permissioned* distributed ledgers (in which membership is vetted, e.g., [9]) is expected to bring benefits to enterprise consortia thanks to the logical centralization of datasets that previously resided at separate sites and were interconnected through various protocols. Prominent examples include financial use cases [27], supply-chain management [22] and provenance [32], to name but a few.

Unfortunately, when deployed in fast networking environments, all current permissioned distributed ledgers exhibit poor performance and are not able to take advantage of the high bandwidth and low latency communication links. For instance, they will take hundreds of milliseconds to commit transactions [9], even if the networking latencies are in the order of microseconds (see Fig. 1). This is because the design of most permissioned distributed ledgers is based on that of public blockchains and, as a result, they are often optimized for wide area networks. For instance, the batching of transactions into blocks is useful for amortizing the high computational cost of Proof-of-Work (PoW) consensus, as well as networking overheads; however, in permissioned distributed ledgers, blocks are a source of added latency and should be removed.

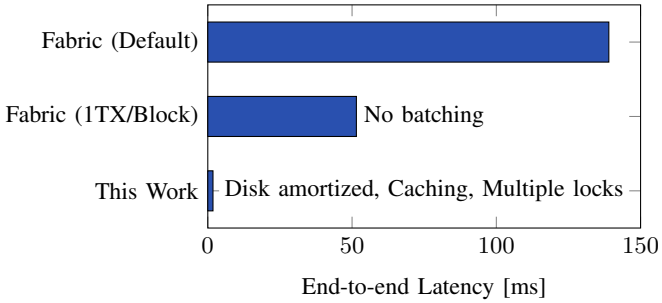


Fig. 1: The latency of Fabric is dominated by block-based batching and disk access. By removing batching and applying further optimizations, latency can be reduced by two orders of magnitude.

In contrast to public blockchains, that assume low bandwidth networking links, permissioned distributed ledgers, similarly to most enterprise software, would typically run in so called approximately synchronous networks [13], [31], that provide reliable low latency and high bandwidth communication both within a single datacenter or across several datacenters. This assumption should also hold in the cloud where all leading providers, such as Amazon [2], Microsoft [7], Oracle [4] and IBM [3], offer hosted enterprise blockchain infrastructures. Even for multi-cloud deployments, there are offerings [6] interconnecting blockchain nodes across multiple clouds using high bandwidth, dedicated, links.

Another fundamental assumption that public blockchains make and design for is the geo-distributed location of nodes. In the permissioned space, even in cross-cloud deployments, wide geo-distribution is often neither required nor beneficial. Stock exchanges, a growing space for enterprise blockchains [1], [5], are characterized by different datacenters and clusters belonging to different providers in close proximity. Since network latencies in such environments will be low and bandwidth high, it is important to ensure that the blockchain technology used to build applications can actually benefit from these to the fullest – which, in turn, motivates the work in this paper

*Work carried out while at the IMDEA Software Institute in Madrid, Spain.

on a streaming distributed ledger (DL), called StreamChain.

In the light of the fast networking assumptions that enterprise blockchain deployments can make, we introduce *latency* as the first-class blockchain performance metric to complement throughput, which has been generally perceived as the main metric of interest [38]. We revisit the design decisions of permissioned distributed ledgers from the perspective of reducing latency and, as a concrete instantiation of our design recommendations, we prototype *StreamChain*, our *streaming variant* of Hyperledger Fabric [9]. In a nutshell, StreamChain departs from block-based system design and chains transactions directly, in a streaming fashion, drastically reducing latency, but without jeopardizing security, reliability or throughput guarantees.

The contributions of this work are as follows:

- *Introducing a latency-efficient permissioned distributed ledger for fast networking environments:* By not batching transactions into blocks and by exploiting the parallelism of modern multicores, StreamChain can achieve end-to-end latencies close to 1.5 ms without compromising throughput. As shown in Figure 1, its latency is two orders of magnitude smaller than vanilla Fabric running on a local area network, and one order of magnitude smaller than Fabric naively configured to create blocks with a single transaction inside. Overall, StreamChain provides latencies that are comparable to traditional databases, thereby making it a more realistic alternative to consider.
- *Showcasing integration with modern programmable hardware accelerators (FPGA-based consensus service):* Even though consensus can be adapted for fast networks [24], [29], current permissioned ledgers do not benefit significantly from such optimizations due to the heavy use of batching. Conversely, latency improvements of consensus make a difference in StreamChain and, as proof-of-concept, we implemented an FPGA-based consensus service that makes it possible to reach commit latencies (i.e., latency excluding smart contract execution) below a millisecond.
- *Platform for future research:* StreamChain is an open-source [1] platform to explore throughput and latency improvements in permissioned ledgers. StreamChain builds on a long-term support (LTS) release of Fabric (v.1.4) which allows future platform improvements to remain compatible with currently existing application ecosystems.

This paper is organized as follows: we provide an overview of permissioned distributed ledgers and Hyperledger Fabric in Section II. The design and implementation of StreamChain is described in Section III. We evaluate the system in Section IV with a YCSB microbenchmark and a Supply Chain Management application. In Section 5, we present a specialized-hardware-based ordering service for StreamChain and explain how we integrated it with software. We talk about the next steps in Section VI and conclude in Section VII.

[1] Source code location removed for double blind reviewing.

II. Background and Related Work

A. Permissioned Ledgers and Fabric

Public ledgers do not authenticate nodes and instead rely on PoW and similar consensus protocols to make Sybil attacks impractical. Permissioned (private) distributed ledgers, on the other hand, authenticate all nodes and therefore allow the use of much cheaper consensus protocols. Nonetheless, permissioned ledgers inherit their design from public ones with many systems evolving from crypto-currency to more general use through "smart contracts" (e.g. Ethereum [39]). Since the execution of the smart contracts has to be serial (Figure 2a), many of these systems are severely limited in throughput when running complex contracts. As a result, Ethereum, for instance, adds a "complexity" fee to smart contracts in the form of "gas", thereby limiting usability in enterprise use-cases.

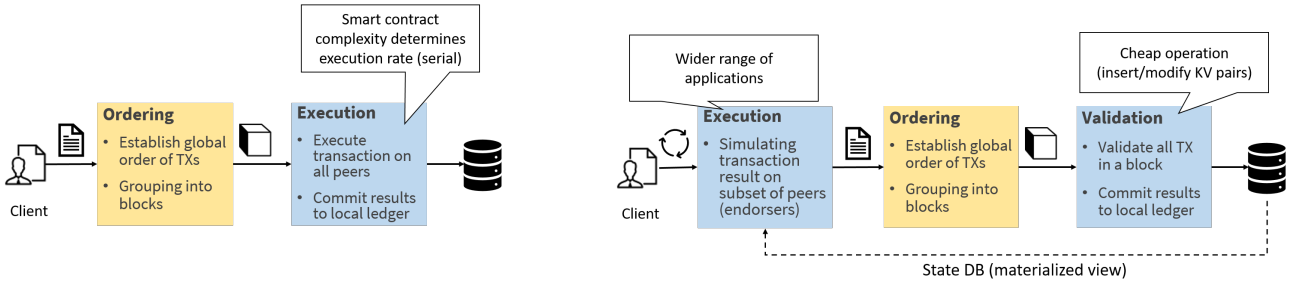
Hyperledger Fabric [9] is an open-source permissioned ledger that allows smart contracts ("chaincode") to be developed in general purpose programming languages. It also removes the execution bottleneck by using an Execute-Order-Validate (EOV) approach (see Figure 2b). As opposed to the Order-Execute (OE) model used, for instance, in Ethereum, where smart contracts are ordered and then shipped to all nodes for execution, in EOV only a subset of nodes execute them. This is done in isolation on top of the "materialized view" of the ledger (Fabric's State DB) to determine their read/write sets and values (TX descriptors). The ordering nodes run a consensus protocol and establish the global order of these read/write sets which are then validated on all peers on their current State DB and committed to the ledger. The EOV model saves compute resources at peers and allows the execution of feature-rich smart contracts.

The drawback of the EOV model is that, since execution and committing happen concurrently on the nodes, it is possible that in the time it takes to order transactions, the underlying state of the variables in the read/write set have changed. This will result in a "failed" transaction whose results are not applied to the ledger state, even though its execution is recorded. This means that in case multiple clients execute chaincode that shares state, the successful transaction throughput (goodput) can be significantly lower than the raw throughput of the system.

B. Improvements and Alternative Designs

Most related work sets out to increase the throughput of permissioned ledgers but end-to-end latencies of single transactions are typically not addressed. However, we believe that this is an important feature to improve, especially in use-cases where network latencies are less than in a wide area deployment.

FastFabric [19] and XOX Fabric [18] bring various optimizations to Fabric which result in an unprecedented 20,000 ops/s throughput. They achieve this through a combination of multi-threaded batch processing, changes to the ordering service and caching deserialized data structures in the validation phase, together with significant changes to the ledger



(a) Order-Execute

(b) Execute-Order-Validate

Fig. 2: Many blockchains, such as Ethereum, process transactions in two steps: establishing global total order first and then executing transactions on each node in order. Fabric processes transactions in three steps, relying on two types of nodes: Peers for Execution and Validations and Orderers to establish a total order of transactions.

and committer pipeline. From their optimisations we have employed caching in StreamChain and reduced, as a result, the cost of the validation step by an additional 10% (see details in Section III-C1).

Sharma et al. [33] focus on increasing goodput in the EOV model and demonstrate that, by relying on database techniques for concurrency control, it is possible to reorder transactions within a batch inside the ordering service in a way that minimizes the number of failing transactions. This idea is applicable to our case as well but it changes an assumption that Fabric makes about the ordering service, namely, that the contents of the transactions are opaque. Depending on the trust model, this raises concerns related to malicious reordering.

CAPER [8] demonstrates how multiple applications that share the same distributed ledger can leverage the fact that they operate on disjoint parts of datasets in order to increase the overall throughput. This is achieved by separating the ordering of operations inside an application from that of ordering across applications. The authors propose a supply chain management workload for benchmarking, similar to the one we use for evaluation.

ResilientDB [20] describes a permissioned blockchain that incorporates a hierarchical consensus protocol design that relies on locality, both in terms of datasets and physical proximity of the nodes, to boost performance. We believe that the streaming approach would be beneficial in such hierarchical designs as well in case the latency improvements on the local execution steps can be translated to end-to-end benefits to clients.

There is an emerging class of distributed ledgers that do not incorporate the notion of blocks at all and operate on a per-transaction basis. Two well known commercial examples that execute without traditional blocks are Corda [11] and IOTA [30]. These systems, however, were designed with a very specific financial services use-case in mind. Instead of storing data in a single chain, a directed acyclic graph (DAG) was used. This allows executing transactions between various subsets of the peers without the burden of global ordering, but also results in systems with different properties than traditional permissioned ledgers.

III. Our solution: StreamChain

A. Target Application Scenarios

StreamChain targets private or public cloud, as well as, multi-cloud deployments, with high bandwidth and low latency networking across nodes (so called "approximately synchronous" [13], [31] environments). Given that, at the moment, virtually all hosted permissioned ledgers are single-cloud, in the current prototype StreamChain tolerates Byzantine peers but uses crash fault tolerant (CFT) ordering, just like Fabric v1.4 LTS. This means that the participants have to trust ordering nodes not to be malicious. In the long run, with the emergence of multi-cloud deployments, it will be necessary to integrate a Byzantine fault tolerant (BFT) ordering service (e.g., [35], [36]) with our streaming optimizations. We discuss the path to achieving this in Section VI.

B. Insights and System Design

StreamChain builds on the following insight: in permissioned ledgers that have fast consensus, blocks have a negative effect on response times in low latency networks. Removing blocks, however, does not change the total amount of computation in the system. It still requires careful parallelization and pipelining of cryptographic operations and disk accesses in order to achieve reasonable throughput. An important side-effect of reducing latency in EOV systems is that it results in higher goodput: lower commit latencies mean less stale data for endorsement, which in turn yields less failed transactions in validation.

In a preliminary workshop version of this work [25] we demonstrated that the main idea behind StreamChain is viable by using a mock implementation that consisted of running Fabric v1.0 configured with 1TX per block and to run in main-memory (i.e., without persistence). In this work we provide a full implementation and further optimizations by:

- re-implementing StreamChain based on Hyperledger Fabric 1.4 LTS in a way that produces a transaction log that remains compatible with vanilla Fabric;
- introducing additional pipelining in the validation stage to increase throughput;

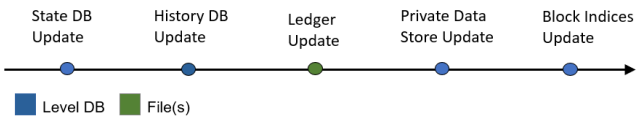


Fig. 3: Several costly disk-accesses are required to commit transactions. Unless disk accesses are batched, their overhead limits throughput severely.

- providing a tunable disk write batching mechanism that ensures persistence while hiding I/O cost; and
- showing that the streaming execution model allows us to further optimize the ordering service, for instance, by including specialized hardware based on FPGAs.

C. Implementation Details

1) Batching and Caching: In Fabric, both the ordering and the validation (commit) stages are writing to disk for each block that is processed in the system, and all transactions, even those that only read from the state, have to be persisted in the ledger. To make matters worse, the commit operation requires several write operations, both to LevelDB and directly to files residing on the disk, as depicted in Figure 3. If we would naively set the size of blocks to a single transaction then latencies would be dominated by the disk flushing cost (see Figure 1) and throughput would be reduced to the IOPS of the underlying device.

In StreamChain, we replace the code-path that flushes the ledger to disk with a local batcher that waits until enough data has accumulated or a time-out is reached (e.g., 100ms) before flushing to disk in an asynchronous manner. To avoid inconsistencies, all buffers are flushed immediately if the code on top performs a read of the underlying data. The same approach applies to all data structures in the peers that are persisted to disk (ledger state, index structures, etc.), with the exception of the materialized view of the ledger that is stored in the State DB.

In Fabric, LevelDB is used by default for the State DB and it persists its own data. The main reason for writing to disk in LevelDB is to speed up recovery in case a node fails and restarts, but we argue that it would be enough to checkpoint the key-value store off the critical path. Hence, we configure LevelDB to run in main memory. Since the ledger is still persisted to disk, persistency is not compromised.

Fabric uses gRPC for the communication between participants of the network and it uses Protocol Buffers to serialize and deserialize messages. This can be an expensive operation with the block data structure that the orderers send to the peers. Deserialization happens selectively and at different levels of granularity; since parts of the block are used several times during the validation phase this leads to inefficiencies. The data structure we use to wrap transactions in StreamChain are very similar to blocks with less overall metadata, and therefore this inefficiency applies to StreamChain as well. As a solution, we implement an idea from FastFabric [19], namely, a cache that stores deserialized block contents to avoid repeated work. Since transactions are received on a single thread and never modified inside the peer, the cache doesn't need to be protected by locks against concurrency.

Beyond the above optimizations, we also modified the way in which the peer reads in its configuration parameters, caching them for subsequent accesses. Furthermore, we disabled an auxiliary data structure, called the history DB that exists to support data provenance queries. Nonetheless, the same batching techniques apply to this data structure as to the main ledger structures.

2) Using Available Parallelism: The peers receive transactions (or blocks in the case of Fabric) from the ordering nodes over the network. These transactions are passed to the Validation logic that performs signature and read/write set checks before it records them in the ledger. In StreamChain we divided the Validation logic into 3 pipeline stages to take advantage of multi-core machines. This layout extends Fabric's two-stage pipeline by one stage and, whereas in Fabric parallel signature checks are only carried out within a block, in StreamChain they work on incoming transactions directly. The pipeline is organized as follows:

- Pipeline Stage 1: Message authentication and endorsement signature check – this step is the most expensive one because it requires checking various signatures to decide whether each transaction has been properly endorsed and ordered. Since there are no data dependencies across transactions, we rely on several cores to parallelize this check and collect the results in a FIFO order. In principle all idle cores of the CPU could be used for this purpose, but we found that in our experimental setup 6-8 were enough to result in pipeline stages with almost equal processing times.
- Pipeline Stage 2: Read/write set check and ledger commit – this step iteratively checks the transactions' read/write sets against the current ledger state (materialized in the State DB) and if no conflicts are found, it commits them to the ledger and updates the State DB. This operation needs to be performed sequentially for transactions so it runs in a single thread. Fabric uses a single reader/writer lock around the State DB and limits parallelism in our pipelined version between endorsement and validation. Therefore, we replace the single lock with a series of locks that allow more fine-grained access.
- Pipeline Stage 3: Additional housekeeping – after updating the ledger state, various additional operations are performed on auxiliary data structures used, for instance, in the gossip protocol. Even though we disable gossip in our deployment (as it's not needed in deployments with plenty of networking bandwidth between ordering nodes and peers), it is important that StreamChain doesn't remove Fabric features. Therefore, these data structure updates are still performed but in their own pipeline stage, which moves them off the critical path.

IV. Evaluation

A. Benchmarking Applications

In the Evaluation of StreamChain, we focus on answering the following questions:

- Can low latency be achieved simply by using blocks of a single transaction?
- What is the relative cost of the different steps and pipeline stages?
- What is the throughput of StreamChain and Fabric with more complex chaincodes?
- Do failing transactions reduce the useful throughput (goodput) of the system significantly?
- Could StreamChain compete with a database?

To answer these questions, we designed two different benchmarks (YCSB and SCM), as described in the following.

1) YCSB-Like Microbenchmark: As a micro-benchmark we relied on the YCSB suite to generate 1KB key-value operations and encoded these as invocations of chaincodes that insert/update/read a value belonging to a key. Experiments were driven by a peer executable that connects to endorsers and invokes the chaincode directly, avoiding an additional RPC from the original Java-based YCSB client. We noticed that, since in the original Fabric version most time is spent on disk writes to update the ledger state, the actual type of operation (set or get) didn't play an important role for performance. We used two workloads: 90% inserts/10% reads (YCSB-90) and equal read/update ratios (YCSB-50).

2) Supply Chain Management Scenario: Since supply chain management (SCM) is an important use-case for distributed ledgers, we designed a benchmark targeting this scenario: vendors order products from each other and update their inventories, provided that they have put a contract in place (n.b. this is not a smart contract, but a condition that allows ordering a specific type of product). The corresponding schema is shown in Figure 4. In addition to transactional queries, such as *create contract*, *place order*, *update order*, etc., two analytical queries are provided: one to compute the *days of supply (local analytics)* and one to compute the *bullwhip coefficient (global analytics)*. The former determines the number of days for which a vendor can supply a product based on current demand and inventory; the latter calculates a number that represents the overall variation between demand and supply across the supply-chain [15].

We implemented all operations as stored procedures in MySQL 8.0 and ran the benchmark from a multi-threaded Java application issuing requests in a closed loop over JDBC. For the distributed ledger version we implemented the queries as chaincode written in Go and organized the key-space by the indexes and primary keys defined in the SQL schema. Data is encoded as JSON within the ledger's key-value pairs. For running the benchmark on Fabric and StreamChain, we used the Java application to export chaincode invocations and we used the same peer executable as for the microbenchmark.

In the supply chain management scenario we focused on the case where contention happens with high probability and, as a result, failing transactions could reduce goodput. To this end, we ran the benchmark with 50 vendors selling 500 products with 6000 contracts between them. Each order issued as part of the benchmark was based on one of these contracts. We varied the portion of transactional queries between 95% (SCM-95) and 99% percent (SCM-99).

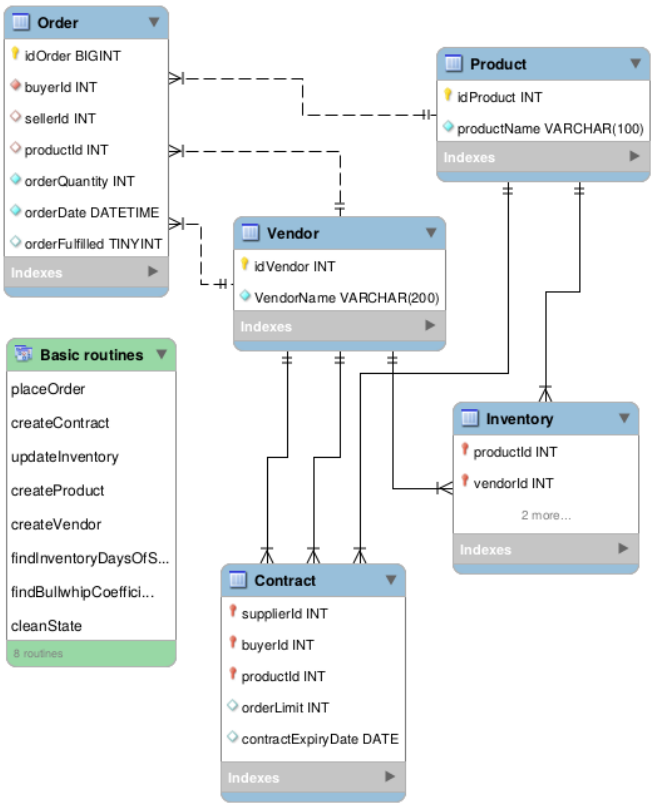


Fig. 4: The SCM example implemented in MySQL consists of tables describing relationships between vendors and keeps track of their orders and current inventories. Both transactional and analytical queries are implemented as stored procedures.

B. Deployment

We ran the experiments on a local 10Gbps cluster of eight servers with Intel Xeon E-2186G CPUs (6 x 3.80GHz), 32GB of RAM and regular HDDs. The blockchain network was composed of 5 peers and 3 ordering service nodes for Raft. We ran MySQL Version 8.0.15 on one of the servers. For benchmarking we used a single client machine with the two workloads described in the previous section and issued 10000 operations not counting data loading, warm-up and cool-down phases (first and last 10% of operations). For benchmarking MySQL, the client was situated on the same machine. Unless otherwise stated we used Fabric with the default block batching of 10 transactions as a baseline.

C. Latency and Throughput

a) Can low latency be achieved simply by using blocks of a single transaction?: To demonstrate that naively setting

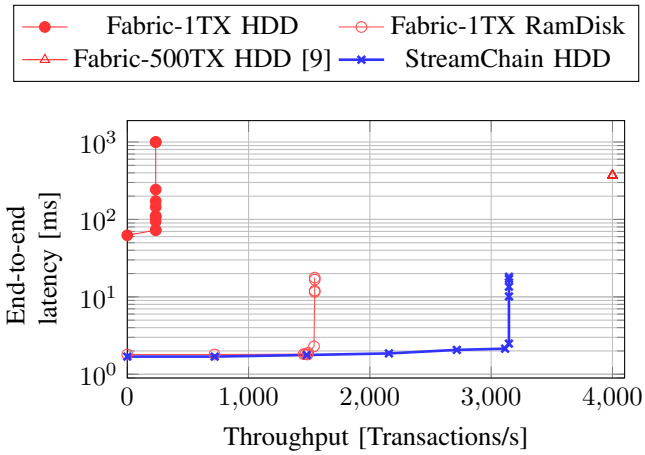


Fig. 5: Fabric with 1TX/Block lowers response times at the cost of throughput. StreamChain offers as low a response time as Fabric run on top of a RamDisk, but 2x better throughput.

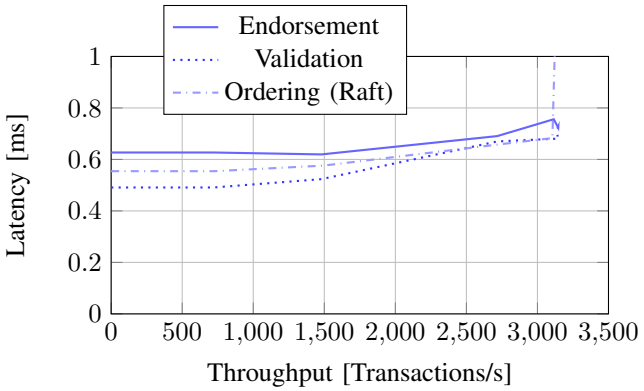


Fig. 6: The main components of the StreamChain pipeline have all predictable latency behavior.

the block size of Fabric to one does not result in great improvements, and hence the optimizations in StreamChain are needed, we show latency as a function of throughput in Figure 5 when using the YCSB-90 workload (we also compare to Fabric in [9], batching 500TXs on average, that uses a comparable micro benchmark). If we set the block size to one (1TX), disk access will dominate both latency and throughput. If, in addition, we "remove" the disk overhead by running on a RamDisk, latency can be lowered significantly but the system becomes non-persistent. In StreamChain we can maintain the low latency behavior and keep persistence, while achieving higher throughput.

b) What is the relative cost of the different steps and pipeline stages?: The latencies inside StreamChain are reduced and balanced across the three stages of the EOV model. Figure 6 shows the breakdown of latency as well as its evolution with increasing load. On our evaluation cluster, the Raft ordering service and the Validation steps become compute-bound at around 3100TX/s. For this reason, in the current design, increasing the throughput of the Ordering service does not immediately translate to end-to-end throughput benefits but it will once the Validation bottleneck is lifted. We discuss one way of achieving this in Section VI.

TABLE I: The SCM benchmark provides a real-world estimate of end-to-end latencies since each query/chaincode performs several reads and writes.

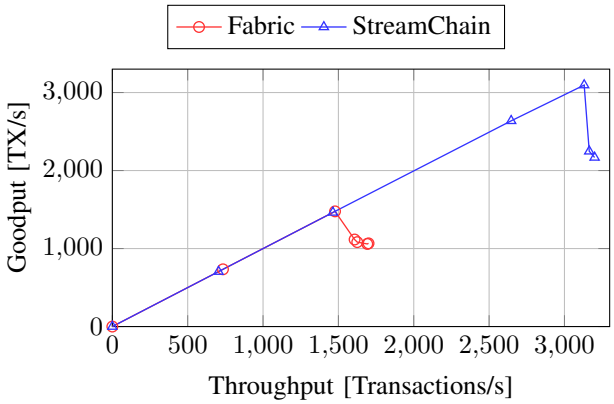
System	SCM-95	SCM-99
Fabric	296.3 ms	169.6 ms
StreamChain	14.8 ms	4.4 ms
MySQL	36.3 ms	37.5 ms

D. SCM Benchmark

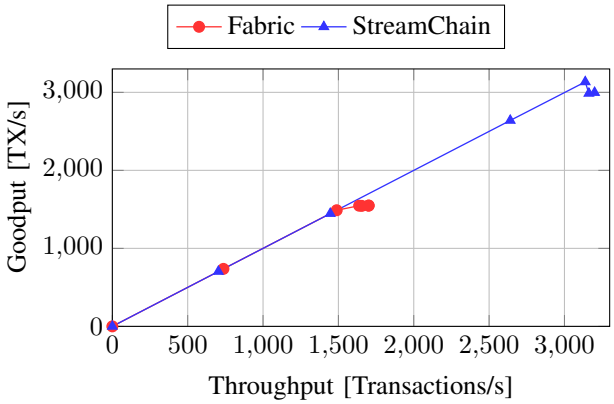
a) What is the throughput with more complex chaincodes?: The SCM workload allows us to measure the latency of the systems with more complex operations. Table I shows the end-to-end latencies of a single client. In the case of Fabric and StreamChain, the latency decreases when the fraction of transactional queries is increased from 95% to 99%. This is expected because analytical queries take longer than transactional ones and often keep a lock on the entire key-space (e.g., in the case of the Bullwhip coefficient) while being endorsed, thereby slowing all validations down. The database exhibits the opposite behavior, whereby each transactional query incurs several write disk accesses, but analytics can be performed on the buffered pages. While there certainly are databases that offer better performance than MySQL, overall, StreamChain delivers competitive latencies.

b) How much do failing transactions reduce the useful throughput (goodput)?: We explore the effect of small working set sizes (more read-after-write contention) on the goodput of StreamChain using the YCSB-50 workload with equal updates and reads and pick the working set as 100%, respectively 10%, of the 10k key space. As shown in Figure 7, Fabric has a negligible number of failing transactions until it reaches 87% of its maximum throughput, where the percentage of failing transactions quickly rises to 40%, reducing the goodput to 60%. Failing transactions are not reissued in this benchmark, but if they were, depending on the backoff policy it could happen that the system is not able to make meaningful progress at this point. Thanks to the reduced data staleness, StreamChain shows a more graceful degradation that happens much later (beyond 97% of its maximum throughput). Even though it cannot eliminate the problem entirely, StreamChain can operate close to saturation with virtually no failing transactions as soon as workloads don't have very small "hotspots".

We also measure goodput with the SCM benchmark. The results, in Figure 8, show two trends: First, by increasing the percentage of transactional queries (from 95% to 99%), the blockchain solutions become faster. Second, both Fabric and StreamChain have a drop in goodput sooner than with the YCSB benchmark. Both trends are to be expected. The endorsement of the analytical operations takes longer and requires locking a large portion of the key-space, thereby slowing down validation. Furthermore, concurrent updates to



(a) High contention: working set of only 1000 key-value pairs



(b) Less contention: working set of 10000 key-value pairs

Fig. 7: When running the YCSB-50 microbenchmark, transactions start failing sooner in Fabric than in StreamChain, which retains almost 100% goodput close to maximum throughput (1700TX/s for Fabric, resp. 3200 TX/s for StreamChain). The difference in behavior is most visible when the number of "hot" key-value pairs being accessed is smaller, resulting in more MVCC contention.

the inventory of vendors result in failing validations, thereby lowering goodput. StreamChain, however, consistently outperforms Fabric.

c) Could StreamChain compete with a database?: when executing the SCM benchmark, MySQL delivers somewhat lower throughput than StreamChain, but of course no failing transactions (in Figure 8 we artificially extend the lines to the right once MySQL reaches maximum throughput). The database is limited by high lock contention and disk flushes for the transactional queries. If we run MySQL on a RamDisk, removing the disk overhead, its throughput increases significantly, to 3000TX/s for SCM-95 and 7000TX/s for SCM-99. These numbers can be regarded as an upper bound for traditional database throughput. Overall, StreamChain can deliver throughput within the range defined by these lower and upper-bounds on a single machine even though it executes in a distributed manner.

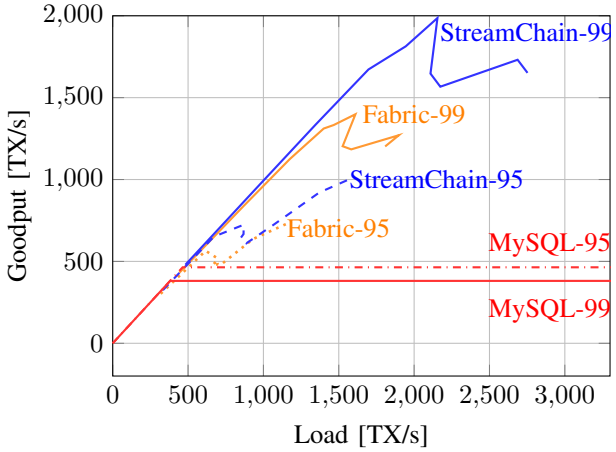


Fig. 8: StreamChain delivers higher goodput than Fabric for the SCM use-case, even though failed transactions become more common with increased load and a higher percentage of analytical queries.

V. Exploring New Opportunities

Since StreamChain exposes the cost of each execution step, it acts as a platform for future exploration and lets us evaluate the usefulness of various emerging hardware features in clouds and datacenters. In this work we focus on the ordering step and show that by incorporating Field Programmable Gate Arrays (FPGAs), which are increasingly available in the cloud, into the ordering service we are able to reduce its latency further and allow it to scale to higher throughput in the future. We target a scenario where the permissioned ledger is hosted by a service provider that has existing infrastructure (e.g. Microsoft Catapult) that can be used to improve the overall performance of the blockchain.

A. FPGAs in the Cloud

Field programmable gate arrays (FPGAs) are hardware chips that are composed of a collection of small look-up tables (LUTs), on-chip memory (BRAM) and specialized digital signal processing units (DSPs), which can be configured and interconnected to implement any hardware circuit. In comparison to traditional processors, FPGAs allow for fine-grained data flow parallelism due to the fact that all "code" is executing in parallel in the device. The energy footprint of FPGAs is an order of magnitude lower than that of server-grade CPUs (and even though it is higher than that of ASICs, FPGAs can be reprogrammed freely, whereas ASICs have fixed functionality). For a more detailed description of FPGA internals and a summary of their strengths in data processing scenarios, we direct the reader to the book by Teubner and Woods [37].

FPGAs are being explored in cloud context both for compute intensive tasks related to machine learning [12] and as low latency key-value store solutions [16], [23], [40] mainly due to their ability to provide predictable, line-rate behavior even when performing near-data processing tasks.

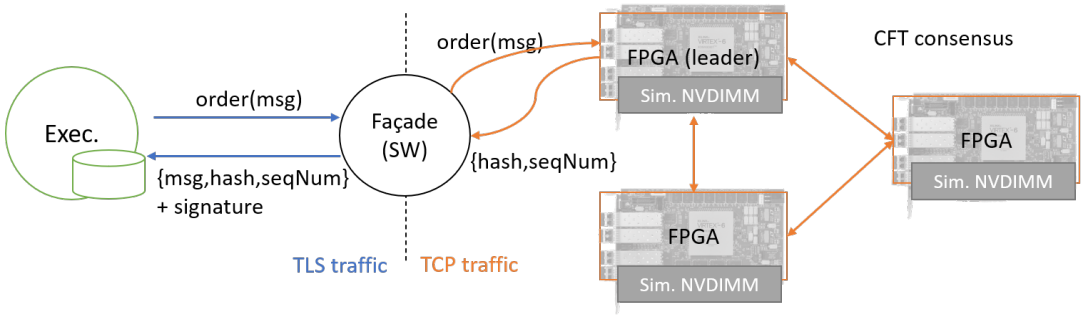


Fig. 9: We integrate a crash fault tolerant ordering service built with FPGAs into StreamChain by using a software node to act as facade between the service and the blockchain peers.

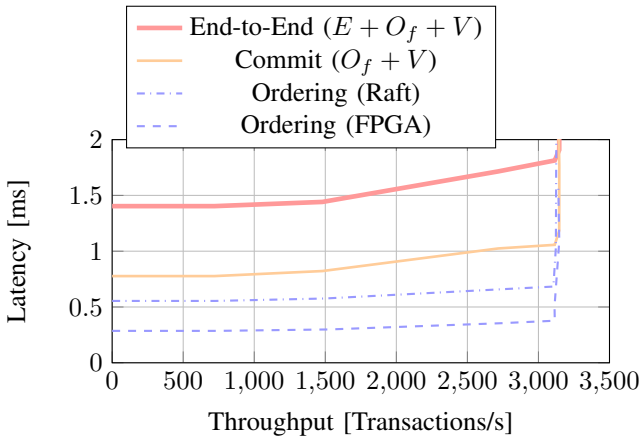


Fig. 10: FPGA-based ordering cuts latency in half when compared to the Raft-based one. Commit latencies with FPGA-based ordering can be as low as 1ms.

Importantly, they are already being used in production for infrastructure acceleration, for instance in Project Catapult [14], to offload networking tasks in Azure virtual machines. With the general availability of FPGAs in the cloud (the number Catapult-enabled Azure servers in 2018 was reported to be more than a million [14]), they could be used to provide functionality packaged as a service with a lower energy footprint, hence lower cost, and more predictable performance than software-based solutions.

B. FPGA-based Ordering

Fabric is designed such that the ordering service can easily be replaced with custom implementations. In version 1.4, it has a service built with Raft [28] as its main way of ordering blocks/transactions. As an alternative, an Apache Kafka based one is available but this introduces higher latencies and more complexity to the system. Fabric also offers a "solo" Orderer comprising of a single node (hence no fault tolerance) that is useful for development and testing purposes.

For the FPGA-based ordering service, we modify Caribou [23], an open source FPGA-based key-value store that incorporates a CFT consensus module for replication [24]. This module implements the Zookeeper's Atomic Broadcast protocol to replicate write operations to other FPGA-based nodes. One of the main benefits of using FPGAs in this context is that they can achieve 10Gbps network-bound performance similar to RDMA-based systems without specializing the network protocol, relying instead on commodity networks and TCP sockets for communication across the consensus nodes and the clients.

a) Design.: Caribou implements a generic put/get interface but for the ordering service a different interaction model is needed. For this reason, we modified the interface and request format of the nodes to expose the following three operations: 1) *order*, that takes a transaction (a BLOB), replicates it across nodes and returns the sequence number assigned to the transaction, 2) *get*, to retrieve the transaction ordered with a specific sequence number and 3) *getLast*, to retrieve the latest ordered transaction and its sequence number. To achieve this functionality, we modified the key-value store implementation on the FPGA to expose the sequence numbers of the consensus algorithm that were hidden before. The design does not rely on blocks and computes the SHA256 hash of each individual transaction inside the FPGA. This hash value is appended to the transactions before they are stored and is returned with *get*s, thereby forming the "links" of the chain.

Our prototyping boards do not have persistent storage, therefore we explored the feasibility of using NVDIMMs for providing durability in the future. We added a module in front of the memory controller on the FPGA to simulate the timings and bandwidth of Intel's Optane NVDIMMs [26]. Given the throughput levels of StreamChain, using lower bandwidth NVMe Flash could also be an option.

b) Integration.: For integration with the rest of the nodes in StreamChain we rely on the code from the "solo" Orderer to act as a facade between peers and FPGAs (Figure 9). It deals with aspects such as TLS flow termination, management messages, etc., and unwraps the transactions and submits them to the FPGA nodes for ordering. For our prototype we used a single facade, but since it is stateless, it could be deployed on multiple nodes to help with fast failover.

c) Evaluation.: We evaluated the FPGA-based ordering service using the same cluster as in Section IV with three Xil-

inx VCU1525 FPGA boards[2] connected to the same 10Gbps switch. These FPGA boards incorporate a large FPGA, similar to that in the Amazon F1 instances, and 64GBs of DRAM. We simulated the NVDIMM behavior on 16GBs worth of memory. Even though the boards are plugged into server machines via PCIe, in this experiment they only rely on the PCIe connection for power.

In Figure 10, we show the latency of the ordering step, as a function of system throughput with Raft and our FPGA-based design. When using FPGAs to run the ordering service instead of Raft, half of its latency (0.3 ms) can be saved. In the current design, our goal was to integrate these nodes without requiring software changes, therefore we opted for a Facade node but in the future latency can be further reduced by replacing this node with in-FPGA TLS termination. Thanks to the pipelined, streaming, design, StreamChain benefits from a lower latency ordering service and achieves commit latencies (Ordering with FPGA + Validation) below 1 ms until saturation. Even end-to-end latencies (Endorsement + Ordering with FPGAs + Validation) remain under 1.5 ms up to almost 2000TX/s.

VI. Future Work

A. BFT Ordering

In order to reduce the necessary trust in a service provider and to ensure that permissioned distributed ledgers can be run in multi-cloud deployments, it is important to have a Byzantine fault tolerant (BFT) ordering service implementation. Using a BFT consensus protocol would also make the system more resilient to arbitrary failures.

Even though there is already work providing BFT ordering in Fabric [35], [36], integration with StreamChain will require modifications to the BFT implementation. This is because batching, used as a default in such protocols [10], [35], [41], has to be eliminated without reducing throughput to impractical levels. An additional challenge is that, since each peer has to be connected to a majority of the BFT ordering nodes to accept transactions as ordered, the jitter across ordering nodes has to be minimized without reducing their ability to sustain high network bandwidth. There are no BFT implementations that fulfill all of the above requirements at the same time because traditionally they have been optimized for geo-distributed operation. Nonetheless, there is already related work that dissects the cost of BFT consensus and identifies latency and throughput bottlenecks, as well as typical sources of jitter [17], [21], [34]. Given these recent efforts in the BFT space, we believe that it will be possible to build a streaming BFT ordering service that can plug into a streaming blockchain solution.

B. Concurrent State Access

Currently, endorsement and validation have to concurrently access the State DB inside endorsing peers. This limits throughput and can increase latencies when locking large portions of the key-space. For this reason, concurrency-enabling optimizations of the State DB will yield the biggest immediate benefits in StreamChain. Of course, fundamentally, the problem of failed transactions cannot be eliminated by changing the State DB. Instead, research is required into a hybrid between the OE and EOV execution models that allows more flexibility in post-order validation.

[2]https://www.xilinx.com/products/boards-and-kits/vcu1525-a.html

VII. Conclusions

In this work we make the case that the design of permissioned distributed ledgers should be revisited for high bandwidth and low latency environments and that latency should be considered as a first-class performance metric. With *StreamChain*, we propose a streaming design and show that it achieves low latency while maintaining high throughput. We demonstrate that StreamChain could be already practical in real-world deployments with a supply chain management benchmark.

Our approach is complementary to ideas for further increasing throughput and, thanks to the elimination of batching overhead, novel research directions open up for hardware-accelerated consensus and cryptography operations in the context of permissioned ledgers. StreamChain is able to take advantage of low latency networking and modern programmable hardware, including emerging platforms such as FPGAs, which, until now, have not been beneficial in such systems.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 842956.

We would like to thank Xilinx for the generous donation of software and hardware used in this work.

References

[1] Australian securities exchange building new blockchain platform. https://www.coindesk.com/australian-securities-exchange-building-new-blockchain-platform-with-vmware-digital-asset, 2019.

[2] Blockchain on AWS. https://aws.amazon.com/blockchain/, 2019.

[3] Blockchain Platform — IBM Cloud. https://cloud.ibm.com/catalog/services/blockchain-platform, 2019.

[4] Blockchain Platform — Oracle Cloud. https://www.oracle.com/application-development/cloud-services/blockchain-platform/, 2019.

[5] Hong kong stock exchange partners with blockchain firm to improve post-trade process. https://cointelegraph.com/news/hong-kong-stock-exchange-partners-with-blockchain-firm-to-improve-post-trade-process, 2019.

[6] Ibm blockchain platform for multicloud. https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/featured_applications/ibm_blockchain_platform.html, 2019.

[7] Microsoft Azure Blockchain. https://azure.microsoft.com/en-us/solutions/blockchain/, 2019.

[8] M. J. Amiri, D. Agrawal, and A. E. Abbadi. Caper: a cross-application permissioned blockchain. *Proceedings of the VLDB Endowment*, 12(11):1385–1398, 2019.

[9] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[10] A. Bessani, J. Sousa, and E. E. Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362. IEEE, 2014.

[11] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn. Corda: An introduction. *R3 CEV, August*, 2016.

[12] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, et al. Serving DNNs in real time at datacenter scale with project Brainwave. *IEEE Micro*, 38(2):8–20, 2018.

[13] A. Desai, S. A. Seshia, S. Qadeer, D. Broman, and J. C. Eidson. Approximate synchrony: An abstraction for distributed almost-synchronous systems. In *International Conference on Computer Aided Verification*, pages 429–448. Springer, 2015.

[14] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, pages 51–66, 2018.

[15] J. C. Fransoo and M. J. Wouters. Measuring the bullwhip effect in the supply chain. *Supply Chain Management: An International Journal*, 5(2), 2000.

[16] E. S. Fukuda, H. Inoue, T. Takenaka, D. Kim, T. Sadahisa, T. Asai, and M. Motomura. Caching memcached at reconfigurable network interface. In *FPL'14*, pages 1–6. IEEE, 2014.

[17] V. Gavrielatos, A. Katsarakis, and V. Nagarajan. Odyssey: The impact of modern hardware on strongly-consistent replication protocols. In *EuroSys '21*, page 245–260, 2021.

[18] C. Gorenflo, L. Golab, and S. Keshav. XOX fabric: A hybrid approach to blockchain transaction execution. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020, Toronto, ON, Canada, May 2-6, 2020*, pages 1–9. IEEE, 2020.

[19] C. Gorenflo, S. Lee, L. Golab, and S. Keshav. FastFabric: Scaling Hyperledger Fabric to 20,000 transactions per second. In *IEEE ICBC*, 2019.

[20] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi. Resilientdb: Global scale resilient blockchain fabric. *Proceedings of the VLDB Endowment*, 13(6).

[21] S. Gupta, S. Rahnama, and M. Sadoghi. Permissioned blockchain through the looking glass: Architectural and implementation lessons learned. In *ICDCS '20*, pages 754–764, 2020.

[22] N. Hackius and M. Petersen. Blockchain in logistics and supply chain: trick or treat? In *Proceedings of the Hamburg International Conference of Logistics (HICL)*, 2017.

[23] Z. István, D. Sidler, and G. Alonso. Caribou: intelligent distributed storage. *PVLDB*, 10(11):1202–1213, 2017.

[24] Z. István, D. Sidler, G. Alonso, and M. Vukolic. Consensus in a box: Inexpensive coordination in hardware. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 425–438, 2016.

[25] Z. István, A. Sorniotti, and M. Vukolić. Streamchain: Do blockchains need blocks? In *SERIAL Workshop at Middleware'18*. ACM, 2018.

[26] J. Izraelevitz, J. Yang, L. Zhang, et al. Basic performance measurements of the intel optane dc persistent memory module. *arXiv preprint arXiv:1903.05714*, 2019.

[27] M. Lokhava, G. Losa, D. Maziéres, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowski, and J. McCaleb. Fast and secure global payments with stellar. In *Proceedings of the 27th Symposium on Operating Systems Principles*, 2019.

[28] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, 2014.

[29] M. Poke and T. Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 107–118. ACM, 2015.

[30] S. Popov. The tangle (2017). *URL https://iota. org/IOTA_Whitepaper. pdf*.

[31] D. R. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 43–57, 2015.

[32] P. Ruan, G. Chen, A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang. Fine-grained, secure and efficient data provenance for blockchain. *PVLDB*, 12(9):975–988, 2019.

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

[34] M.-K. Sit, M. Bravo, and Z. Istvan. An experimental framework for improving the performance of bft consensus for future permissioned blockchains. In *DEBS '21*, 2021.

[35] J. Sousa, A. Bessani, and M. Vukolić. 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*, pages 51–58, 2018.

[36] C. Stathakopoulou, T. David, M. Pavlovic, and M. Vukolić. Mir-bft: High-throughput robust bft for decentralized networks. *arXiv preprint arXiv:1906.05552*, 2019.

[37] J. Teubner and L. Woods. *Data Processing on FPGAs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.

[38] M. Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, pages 112–125, 2015.

[39] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.

[40] S. Xu, S. Lee, S.-W. Jun, M. Liu, J. Hicks, et al. BlueCache: A scalable distributed flash-based key-value store. *PVLDB*, 10(4):301–312, 2016.

[41] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.