Scientific Research Publishing

Journal of Computer and Communications, 2025, 13(10), 49-64
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

SmartFlow: A Lightweight Protocol for Fine-Grained Sharding and Parallel Smart Contract Execution in High-Performance Blockchain Systems

Amena Begum Farha, Abdullah Al-Mamun

School of Computer and Cyber Sciences, Augusta University, Augusta, USA
Email: afarha@augusta.edu, aalmamun@augusta.edu

How to cite this paper: Farha, A.B. and Al-Mamun, A. (2025) SmartFlow: A Lightweight Protocol for Fine-Grained Sharding and Parallel Smart Contract Execution in High-Performance Blockchain Systems. *Journal of Computer and Communications*, **13**, 49-64.
https://doi.org/10.4236/jcc.2025.1310003

Received: July 24, 2025
Accepted: October 8, 2025
Published: October 11, 2025

Abstract

Blockchain technology and smart contracts have gained considerable attention across various industries due to their transformative potential. However, traditional blockchain protocols face significant challenges in high-performance computing (HPC) environments, particularly in terms of resource consumption and energy efficiency. This paper introduces **SmartFlow**, a novel framework aimed at enhancing blockchain performance in HPC environments. SmartFlow features two core modules: 1) a fine-grained transaction sharding graph mechanism that enables improved concurrent transaction processing, and 2) a parallel smart contract execution system designed to handle complex blockchain transactions efficiently. Both modules are integrated with a lightweight two-phase quorum commit (2PQC) protocol, optimized using a dual-layer parallel execution strategy, which significantly improves the scalability and performance of blockchain systems. The framework is implemented and evaluated in an HPC cluster, demonstrating promising results with up to 7.5× lower latency and 5.8× higher throughput compared to state-of-the-art blockchain protocols.

Keywords

Blockchain, Smart Contract, Consensus Protocol, Sharding, Parallelization, High-Performance Computing

1. Introduction

1.1. Motivation

Blockchain technology has demonstrated transformative potential across various

industries [1]. In high-performance computing (HPC) environments, it offers significant advantages, such as distributed consistent caching, data fidelity, and provenance tracking [2]-[5]. Smart contracts further enhance blockchain's capabilities by automating complex workflows, reducing reliance on intermediaries, and optimizing computational costs [6]. However, achieving high-throughput blockchain execution in HPC settings requires addressing critical limitations inherent in traditional blockchain protocols.

A major bottleneck in existing blockchain systems is their reliance on heavyweight consensus mechanisms like Proof of Work (PoW) [7] and Practical Byzantine Fault Tolerance (PBFT) [8]. These methods impose high computational and energy costs while introducing significant latency, hindering efficient large-scale execution. Additionally, mainstream blockchain architectures are incompatible with the Message Passing Interface (MPI)—a widely adopted framework in the scientific computing community for HPC workloads. Since MPI facilitates distributed memory processing and efficient inter-node communication, the absence of blockchain systems designed to integrate with it further limits their feasibility in HPC environments—particularly for parallel transaction processing, where thousands of smart contracts may need to be executed concurrently.

Another major limitation is that most blockchain implementations *lack compatibility with shared storage frameworks*, which are fundamental to modern HPC and scientific computing workflows [2]. Traditional blockchain architectures rely on independently maintained ledgers across nodes, making integration with HPC storage solutions—such as parallel file systems or distributed shared memory—challenging. Furthermore, despite the automation advantages of smart contracts, their execution remains predominantly sequential in many blockchain frameworks. This sequential processing creates significant performance bottlenecks when handling high-volume workloads, limiting blockchain's applicability in large-scale scientific and HPC applications.

To address these limitations, the blockchain for HPC requires: 1) a *lightweight consensus mechanism* that minimizes overhead while maintaining security and resilience, 2) *efficient parallel execution of smart contracts* across a large scale without contention or synchronization delays, and 3) *seamless integration with existing HPC infrastructure*, including shared storage and MPI-based communication frameworks.

1.2. Challenges

Integrating a lightweight blockchain framework into HPC systems to support parallel smart contract execution presents several technical challenges that must be addressed to ensure seamless scalability, efficiency, and fault tolerance. The primary challenges include the following.

Lightweight Consensus Protocol: Existing blockchain consensus mechanisms impose significant computational and synchronization overheads, limiting their suitability for high-speed and high-volume transaction environments. A light-

weight and parallel-friendly consensus protocol compatible with shared storage ecosystems and parallel smart contracts is required to enable efficient block processing, low latency consistency, and high fault tolerance while remaining compatible with shared storage platforms.

Parallel Execution of Smart Contracts: Scheduling smart contracts across distributed compute nodes introduces data dependencies, conditional logic constraints, and state synchronization challenges [9]. Unlike conventional parallel computing, blockchain execution requires strict ledger consistency, meaning transactions must be validated concurrently without violating causal dependencies. Optimizing energy consumption, concurrency control, and rollback mechanisms is critical to ensuring scalable execution [10].

Integration with MPI and Technical Challenges: Integrating decentralized blockchain networks with scientific computing frameworks such as the Message Passing Interface (MPI) introduces several critical bottlenecks that need to be addressed. Efficient consensus formation across distributed nodes requires precise synchronization strategies to prevent network congestion while maintaining blockchain immutability. Balancing the fast, mutable memory demands of HPC with the immutability of blockchain transactions is a significant challenge. Additionally, optimizing block timing, propagation latency, and transaction finalization is crucial for achieving high throughput. A well-designed consensus mechanism must minimize synchronization overhead during parallel validation, ensuring fault tolerance and security at scale. Moreover, smart contract modularization should enable decentralized execution without introducing bottlenecks in the verification process, ensuring the system operates efficiently in large-scale environments.

1.3. Contributions

This paper introduces SmartFlow, a lightweight mechanism for blockchain that enables parallel execution of smart contracts, addressing the performance bottlenecks associated with traditional sequential execution methods [11]. Unlike conventional resource-intensive protocols [12], the proposed protocol is designed for improved efficiency with minimal resource consumption, enabling simultaneous smart contract processing. This enhances system performance, mitigates the limitations of traditional consensus approaches, and positions it as a significant advancement in scalable smart contract execution within blockchain systems.

In summary, this paper makes the following contributions.

• We design a fine-grained transaction sharding graph mechanism to enhance the parallel processing of transactions, thus improving system efficiency by optimizing resource usage and throughput.

• We develop a parallel smart contract execution framework, guided by a lightweight two-phase quorum commit (2PQC) protocol [2], which enables complex contract logic to be processed in parallel across distributed nodes. This approach improves both efficiency and consistency in smart contract processing.

• We optimize the lightweight two-phase quorum commit (2PQC) protocol, integrated with MPI, utilizing a dual-layer strategy to address communication overhead and throughput limitations, providing a scalable solution for blockchain systems in HPC environments.

• We implement a prototype and evaluate the system using a real-world dataset (e.g., banking transactions) [13] and compare its performance against RapidChain, a mainstream blockchain protocol, and a cutting-edge two-phase concurrency control (2PCC) protocol for parallel smart contract execution [14]. Our preliminary experiments show that the proposed system offers up to 7.5× lower latency and 5.8× higher throughput compared to current state-of-the-art protocols.

2. Related Works

On a blockchain, smart contracts are self-executing units of code that automate contract execution according to predetermined conditions. Designed with programming languages such as Solidity, they function independently, carrying out tasks without the need for a third party. The exploration of serial and parallel execution paradigms has yielded noteworthy advances. Chaincode introduces a scalable framework leveraging serial execution with state-of-the-art consensus protocols [15] and efficient data structures. Čeke *et al.* [16] focus on optimizing serial execution by reducing gas costs and improving transaction processing times.

Parallel execution in blockchain has become a crucial research direction to improve the performance and scalability of smart contract execution. XuperChain [17] employs parallelization without specifying the consensus protocol, while Liu *et al.* [18] separate execution from consensus in Ethereum to boost throughput. C. Jin *et al.* [14] introduce a two-phase concurrency protocol for permissioned blockchains, optimizing validation and execution stages. Chung and Park [19] propose parallel sharded execution using shared logs to address conflicts and bottlenecks in Ethereum's Order-Execute model. Fang *et al.* [20] demonstrate gains in internode performance with SEFrame, which uses Intel SGX enclaves for safe parallel execution. Tao *et al.* [21] improve blockchain throughput and performance with a distributed sharding approach.

Other works focus on privacy and accessibility. Li *et al.* [22] prioritize privacy through contract segmentation, while Tan *et al.* [23] present LATTE, a visual interface for non-programmatic smart contract creation. Jian *et al.* [24] explore hardware-based security techniques with TSC-VEE, a TrustZone-based virtual execution environment. Qi *et al.* [25] introduce DMVCC, a scheduling framework for fine-grained synchronization in parallel execution. Despite these advancements, as shown in **Table 1**, many studies still rely on resource-intensive consensus protocols such as Practical Byzantine Fault Tolerance (PBFT), generic Byzantine Fault Tolerance (BFT), or Proof of Stake (PoS), which are impractical in high-performance computing (HPC) environments. Furthermore, none of these protocols support shared storage ecosystems or Message Passing Interface (MPI) based parallel computation.

Table 1. Summary of limitations of recent Smart Contract Protocols.

Features	A High Performance Concurrency Protocol [14]	Parallel and Asynchronous Smart Contract Execution [18]	SEFrame [20]	Parallel Execution of Solidity Smart Contract [19]	DMVCC [25]	SmartFlow (This Work)
Lightweight Computation	×	×	×	×	×	√
Energy-efficient Protocol	×	×	×	×	×	√
Cost-effective Communication	×	×	×	×	×	√
Parallel block processing	√	×	×	×	×	√
MPI-compliant	×	×	×	×	×	√
Shared-storage compatible	×	×	×	×	×	√

Recent developments in MPI provide extensive knowledge to improve its properties to create various solutions [26] [27]. However, MPI has not been used to develop a consensus protocol for parallel smart contract execution. Integrating MPI with existing research enhances MPI-specific packages. While a recent work [2] has proposed lightweight protocols to enhance blockchain compatibility with HPC ecosystems, it does not provide support for parallel execution of smart contracts.

3. System Design

The architecture of our proposed system, SmartFlow, consists of four main components: the Transaction Sharding Graph, Smart Contract, SmartFlow Consensus Protocol, and Parallel Processing component. As illustrated in **Figure 1**, the transaction processing flow begins with the Transaction Sharding Graph, which analyzes dependencies and organizes independent transactions into batches. These batches are then transmitted to the blockchain network, composed of MPI-connected nodes. Each node participates in a two-phase validation and quorum commit consensus protocol. First, transactions in each batch are validated in parallel by executing their associated smart contracts across the available cores. If validation succeeds, contract state updates are provisionally applied. After parallel execution, the nodes vote to commit or abort the batch based on validation results. If a quorum is reached, the block is committed to the chain. This validate-then-commit approach, leveraging parallel execution, aims to accelerate transaction latency and throughput by maximizing concurrent validation and contract execution across available HPC resources. We will discuss each component in more detail in the following sections.

3.1. Transaction Sharding Graph

The **Transaction Sharding Graph (TSG)** is a fundamental component of our pro-

posed system architecture, designed to efficiently manage and analyze dependencies between transactions. Structured as a *directed acyclic graph* (*DAG*), each node in the TSG represents an individual transaction, while directed edges indicate dependencies between transactions. The direction of an edge enforces dependency constraints, ensuring that the graph remains acyclic and that transactions are executed in a valid sequence.

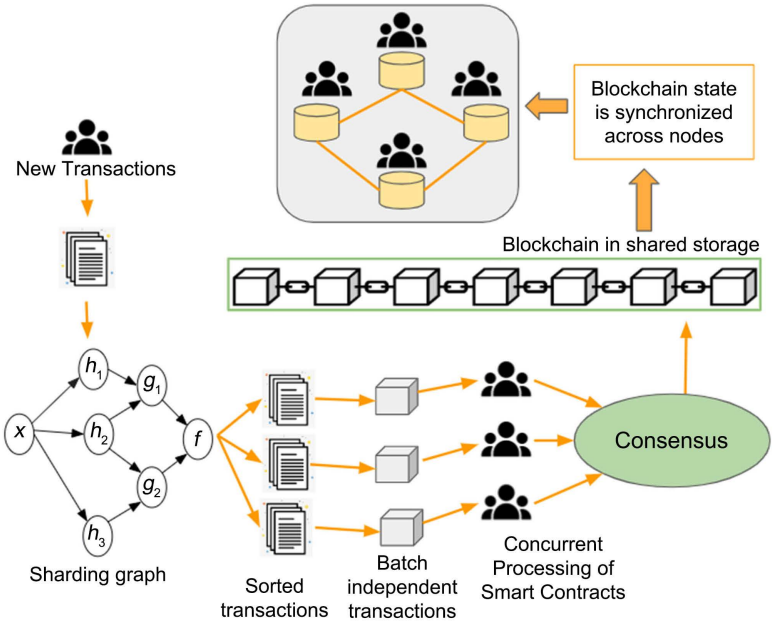

Figure 1. Overall system architecture of the proposed protocol.

Figure 2 illustrates an example of a Transaction Sharding Graph (TSG), where nodes represent transactions, and directed edges capture dependency relationships. The execution order within the TSG ensures that transactions are processed only after their dependencies are resolved. In this example, the execution sequence $[T_1, T_7, T_8, T_3, T_6, T_4, T_2, T_5]$ guarantees correct processing while maximizing parallelism. After ordering transactions based on their execution dependencies, independent transactions are grouped into batches and distributed to separate shards, each managed by a set of nodes.

The process described in Protocol 1 begins by iterating through the list of transactions T. Each transaction T_i is checked for independence. If it has no dependencies, a hash is created using the corresponding entity E, and the transaction is added to a batch B. Dependent transactions are deferred until their required conditions are met (Lines 1 - 9). Once the batch is prepared, it is partitioned into blocks based on the number of available clusters. Each block b_i is then assigned to a corresponding cluster C_i. Finally, the protocol invokes the consensus mechanism to validate each block in parallel across the assigned clusters (Lines 10 - 22). By organizing transactions into shards, the TSG enhances parallelism and computational efficiency. Additionally, the directed acyclic structure of the graph prevents cyclic dependencies, ensuring consistent and reliable transaction execution.

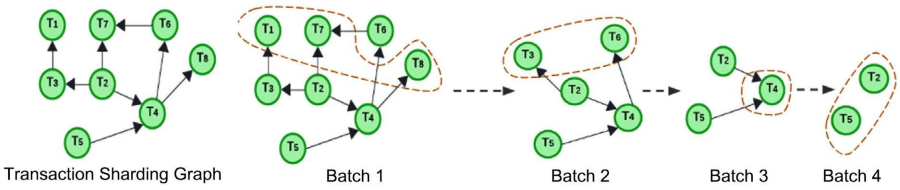

Figure 2. Generating batches of independent transactions through transaction sharding graph.

Protocol 1 Transaction Sharding Graph

Require: Clusters $C = \{C_1, C_2, \ldots, C_n\}$; Commits $T = \{T_1, T_2, \ldots, T_m\}$; Entities $E = \{E_1, E_2, \ldots, E_m\}$; Batch B; Blocks $b = \{b_1, b_2, \ldots, b_k\}$
Ensure: Parallel processing of independent transactions in T

function PROCESS_TRANSACTIONS(C, T, E)
  for $T_i \in T$ **do**
    if T_i is independent **then**
      Create hash H_{T_i} using entity E_i
      Inject T_i into Batch B
    else
      Wait until all dependent transactions are resolved
    end if
  end for
  DIVIDE_BATCH(C, B, b)
  CONSENSUS(b, C, E)
return
end function
function DIVIDE_BATCH(C, B, b)
  $k \leftarrow |C|$ ▷ Number of clusters
  $n \leftarrow |B|$ ▷ Number of transactions in the batch
  $block\_size \leftarrow \lceil n/k \rceil$
  for $i = 1$ to k **do**
    $b_i \leftarrow B[(i-1) \cdot block\_size + 1 : i \cdot block\_size]$
    Assign b_i to cluster C_i
  end for
return b
end function

3.2. Smart Contract

The smart contract module in our system plays a critical role in executing business logic and ensuring transaction validity. Implemented in Python, the smart contract handles key banking functions, such as depositing funds, transferring money, and withdrawing funds. More importantly, the smart contract is designed to be invoked during the parallel validation and execution phase managed by the consensus protocol.

A key technical contribution of our approach is the integration of the *Transaction Sharding Graph* (*TSG*), which enables parallel transaction processing by analyzing dependencies among transactions and partitioning them into independent execution batches. Each batch contains transactions that can execute concurrently, reducing bottlenecks and increasing throughput. These batches are then processed in parallel by the smart contract module deployed in the distributed nodes utilizing the *Parallel Processing* component. By leveraging the TSG, the smart contract efficiently validates dependencies and ensures that updates to contract states remain consistent across the network. Once validated, the smart contract's

changes are temporarily applied, awaiting consensus confirmation. This integration not only synchronizes contract execution with the consensus process but also optimizes distributed transaction processing. As a result, the smart contract contributes to both the *SmartFlow Consensus Protocol* and the *Parallel Processing* components, achieving high transaction throughput while preserving consistency and security within the system.

3.3. SmartFlow Consensus Protocol

In the proposed system, a lightweight MPI-based Quorum Commit protocol is utilized to overcome the limitations of traditional protocols (e.g., PBFT, PoW, PoS). This protocol is specifically designed for distributed blockchain networks, where the root node serves as the coordinator. The coordinator manages a subcluster of participant nodes and oversees transaction validation and commit operations. The workflow of the proposed Two-Phase Quorum Commit (2PQC) protocol is illustrated in Figure 3. Upon receiving a batch of transactions from the Transaction Dependency Graph, the primary node processes them through the 2PQC protocol, which operates as follows:

Prepare Phase: The coordinator initiates the process by sending a prepare message along with the set of transactions to all participant nodes. Each participant node independently validates the transactions based on the associated smart contract and responds with a prepare vote, indicating readiness or non-readiness to commit.

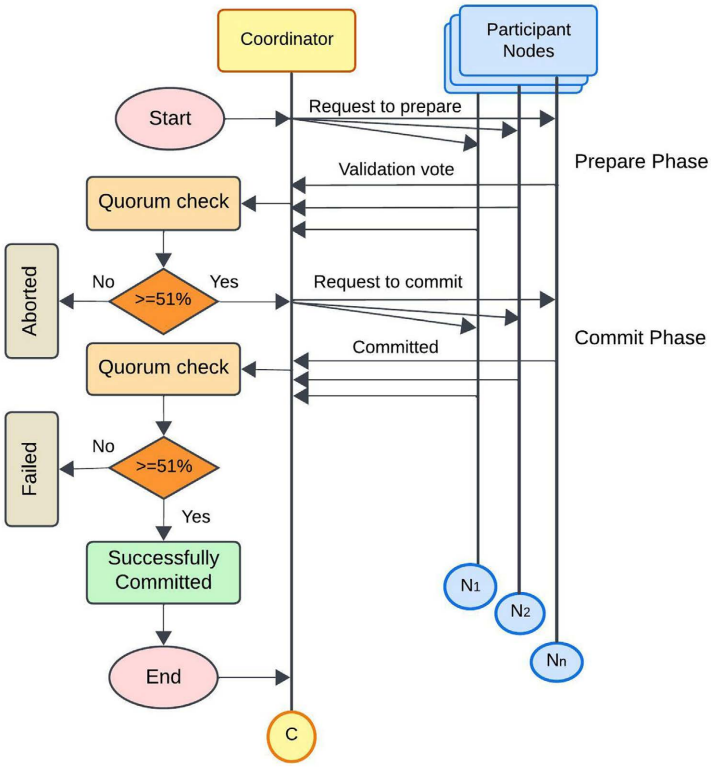

Figure 3. Customized SmartFlow Protocol for Enhanced Consensus Mechanisms.

Protocol 2 SmartFlow Consensus Protocol

Require: Clusters $C = \{C_1, C_2, \dots\}$, where each C_i has a sub-cluster N_i with n nodes and a coordinator N_{i_c}; Entities $E = \{E_1, E_2, \dots\}$; Transactions $T = \{T_1, T_2, \dots, T_m\}$ batched into blocks b; Each block is handled by a cluster coordinator; Hash list blh for b

Ensure: At least 51% of nodes in C agree to validate block b, stored in $agreedNodes$

```
1:  function CONSENSUS(b, C, E)
2:      for each cluster C_i ∈ C do
3:          Assign block b_i to C_i
4:          for each transaction T ∈ b_i do
5:              Parse metadata from T
6:              Extract security metadata (e.g., account, operation)
7:          end for
8:          for each node N_i ∈ C_i do
9:              agreedNodes ← VALIDATION(b_i, N_i, E)   ▷ Validate block b_i with entity E
10:         end for
11:     end for
12:     if |agreedNodes| > n/2 then
13:         Commit b_i
14:     else
15:         Abort b_i
16:     end if
17: end function
```

Quorum Check: Following the prepare phase, the coordinator conducts a vital quorum check. If the majority of participant nodes (*i.e.*, 51% or more) provide validation votes, it proceeds to request a commit. If not, the transaction block is aborted. The quorum check in 2PQC ensures that the coordinator only commits transactions when a majority of nodes agree, preventing inconsistent or invalid states.

Commit Phase: After a successful quorum check, the coordinator issues a commit request to all the participant nodes. Each node then commits to the transactions and sends a confirmation message back to the coordinator. Once all confirmation messages are received, the coordinator performs another quorum check.

Final Decision: Based on the results of the quorum check in the commit phase, the coordinator makes the final decision. If it receives commitments from at least 51% of the participant nodes, it declares the transactions committed successfully. Otherwise, it concludes that the commitment has failed.

As detailed in Protocol 2, the SmartFlow Consensus Protocol commences by distributing transaction block b_i to each cluster C_i, where a sub-cluster N_i with n nodes is managed by a coordinator N_{i_c}. Each transaction is parsed to extract metadata and security information (Lines 1 - 7). Nodes in the sub-cluster independently validate the block using entity data E (Lines 8 - 11). If more than 51% of the nodes agree, the block is committed; otherwise, it is aborted (Lines 12 - 16).

In our HPC blockchain system, where all nodes are trusted and pre-authorized, Byzantine faults are considered unlikely, making a 51% majority quorum sufficient for commit decisions—contrasting with the 2/3 majority required in typical zero-trust blockchain environments. Transactions are only finalized after achieving quorum consensus across nodes also this ensures consistency even in cases of partial failures or temporary network partitions. This protocol ensures that isolated nodes or failed nodes cannot commit invalid states during any node failures

or network partitions since they cannot reach the required quorum. When these nodes reconnect, the states are synchronized via the quorum-based commit process. This synchronization works for consistency and for network integrity. By leveraging a decentralized verification service, the coordinator's actions are transparent and verifiable. Each commit to the blockchain is hashed and recorded on an immutable ledger, allowing contributors to independently verify updates. Any discrepancies are flagged through a protocol, ensuring unauthorized changes are prevented. This structure ensures the coordinator's actions are accountable to all participants, maintaining trust.

3.4. Parallel Processing: Dual-Layer Strategy

To optimize parallelism in an MPI-based blockchain framework, we introduce a *dual-thread execution strategy* that assigns two dedicated threads per node: one for *transaction validation* and another for *consensus and commit handling*. This division ensures efficient utilization of computational resources while minimizing inter-thread contention. Within each MPI node, the *validation thread* is responsible for verifying incoming transaction batches. Transactions are processed in parallel or micro-batches within this thread, ensuring correctness while maximizing throughput. Meanwhile, the *consensus/commit thread* asynchronously communicates validated transactions with other nodes using *non-blocking MPI primitives* (MPI_Isend, MPI_Irecv), allowing consensus operations to proceed without stalling validation.

This model effectively decouples validation from the consensus process, ensuring that while one thread verifies transactions, the other can finalize *commits* and synchronize state updates across nodes. By leveraging *overlapping execution*, where validation continues while previous transactions await consensus, the system significantly reduces bottlenecks. Furthermore, this approach reduces the complexity of thread management while maintaining high scalability. The consensus thread can aggregate multiple validation results before reaching a commit decision, thereby minimizing MPI communication overhead. By employing a *minimal yet effective dual-thread model*–synergizing thread-level and data-level parallelism—our approach optimally utilizes computational resources, significantly accelerating transaction validation and contract execution, making it highly suitable for *HPC environments* requiring high transaction throughput with minimal synchronization delays. Although HPC environments are often associated with centralized architectures, using HPC infrastructure for blockchain does not inherently compromise decentralization. In our design, decentralization is preserved by ensuring that each MPI node independently validates transactions and interacts equally in the consensus process, rather than relying on a single central node for decision-making. Our system leverages parallel storage systems in HPC environments, allowing MPI nodes to efficiently access and synchronize data across the distributed network. This integration of parallel storage with independent transaction validation upholds the principles of decentralization while benefiting from the perfor-

mance advantages of HPC resources.

4. Evaluation and Results

4.1. Experimental Setup

4.1.1. Testbed

The experiments are conducted on an HPC cluster with 18 compute nodes (2× Intel Xeon Silver 4210R, 20 cores, 96 GB RAM) connected via an Infiniband network to a 240 TB shared storage managed by Lustre [28]. Each node runs Red Hat Enterprise Linux 8, OpenMPI, MPI4PY, Python 3.12.2, and Numpy 1.19.5. Each compute node can emulate up to 20 virtual nodes using user-level threads. To achieve optimal performance in a time-shared system, we deploy the prototype across cluster sizes ranging from 10 to 100 nodes.

4.1.2. Workload

For our experiments, we use the financial transaction dataset [13] with varying transaction volumes, ranging from small batches of 2000 transactions to larger volumes of 30,000 transactions. We specifically leverage banking transactions because the algorithms used to validate and process these transactions are well-suited for optimizing large-scale simulations of economic systems and parallel data processing in scientific computing (e.g., climate modeling). These workloads involve complex transaction dependencies, requiring runtime analysis and parallel processing to enhance performance.

4.1.3. Systems for Comparison

As our first baseline, we deployed a two-phase concurrency control (2PCC) protocol with an optimized PBFT (Practical Byzantine Fault Tolerance) protocol for parallel smart contract execution [14], aligning closely with SmartFlow's strategy. Our second baseline was RapidChain [29], a recognized blockchain protocol that also uses PBFT, but with the vanilla smart contract execution method. Both 2PCC and RapidChain focus on Byzantine fault-tolerant consensus and smart contract execution. This contrast allows evaluation of our protocol's performance against parallel and conventional execution models under Byzantine settings. This setup allowed for a direct performance comparison under the same operational conditions. We excluded proof-of-stake (PoS) protocols due to their susceptibility to centralization and other attacks such as the Sybil attack [30] [31]. The evaluation aimed to evaluate the efficiency and effectiveness of SmartFlow compared to these baselines, focusing on their ability to execute smart contracts effectively at scale within an HPC cluster.

4.2. Latency

We compared the latency of the SmartFlow framework against two state-of-the-art blockchain systems: 2PCC Protocol (Two-Phase Concurrency Control) and RapidChain. Latency refers to the time required to process a varying number of transactions. The experiment was conducted on a 50-node blockchain network.

Figure 4 presents the transaction processing performance of the three systems as the number of transactions increases from 2000 to 10,000. SmartFlow consistently outperforms both 2PCC and RapidChain. At 2000 transactions, it is approximately 3.9× faster than 2PCC and 5.7× faster than RapidChain. At 10,000 transactions, SmartFlow remains efficient, being 2.1× faster than 2PCC and 3.4× faster than RapidChain, demonstrating its scalability under higher workloads.

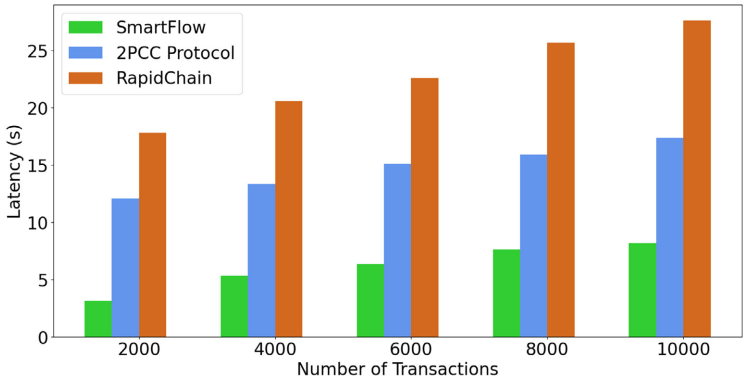

Figure 4. Latency comparison of SmartFlow and state-of-the-art blockchain systems on a 50-node cluster.

4.3. Throughput

In this section, we evaluated the throughput of SmartFlow in comparison to the 2PCC (Two-Phase Concurrency Control) protocol and RapidChain, measured in transactions per second (TPS). The experiment maintains a constant workload of 1000 transactions while varying the number of nodes from 10 to 50. Figure 5 illustrates the TPS performance of SmartFlow alongside the other blockchain systems. SmartFlow repeatedly surpasses the conventional systems across all node scales, achieving up to 4.4× and 5.8× higher throughput than 2PCC and RapidChain, respectively. Thanks to its optimized 2PQC (Two-Phase Quorum Commit) consensus protocol, SmartFlow's throughput scales efficiently, even at larger network sizes, whereas both 2PCC and RapidChain experience performance degradation as the number of nodes increases.

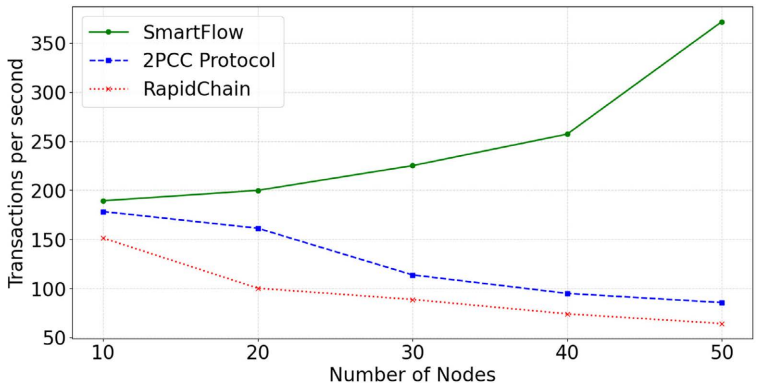

Figure 5. Throughput comparison of SmartFlow and state-of-the-art blockchain systems on a 50-node cluster.

4.4. Scalability

This section evaluates the scalability of SmartFlow by measuring its performance under increasingly large workloads. We assess how the system's latency scales with growing transaction volumes in an HPC environment, comparing it to the 2PCC (Two-Phase Concurrency Control) protocol and RapidChain. Figure 6 illustrates the observed latency, measured in seconds, across all systems. Throughout this experiment, we utilized a network of 100 nodes and varied the number of transactions from 10,000 to 30,000. It is noteworthy that SmartFlow significantly outperforms the baseline systems at all transaction scales, achieving latency up to 3× and 7.5× faster than the 2PCC protocol and RapidChain, respectively. As the workload increases, SmartFlow sustains its performance advantage, delivering increasingly efficient results compared to other blockchain systems. The combination of the dual-layer parallel execution strategy and the lightweight 2PQC consensus protocol effectively manages the growing workload, reducing latency and improving overall performance across the distributed nodes.

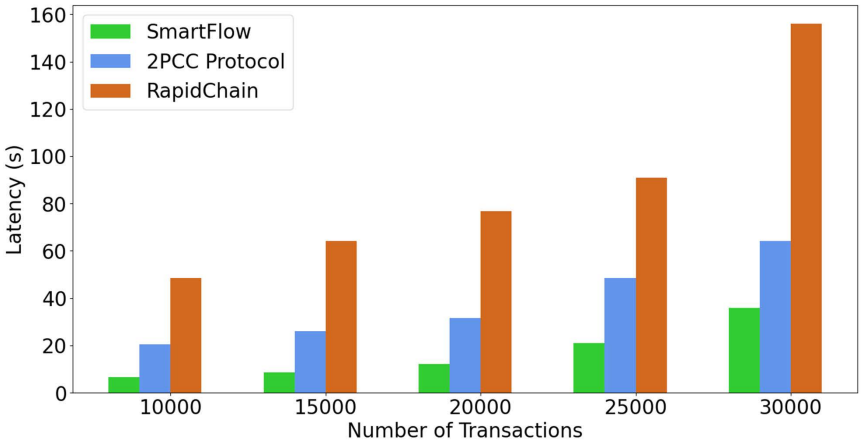

Figure 6. Scalability analysis of SmartFlow and state-of-the-art blockchain systems in a 100-node HPC cluster.

5. Conclusion and Future Work

SmartFlow introduces two key innovations to enhance blockchain performance in high-performance computing (HPC) environments: 1) a fine-grained Transaction Sharding Graph (TSG) for maximizing concurrency in transaction processing and 2) a parallel smart contract execution model compatible with MPI, integrated with a lightweight Two-Phase Quorum Commit (2PQC) protocol optimized for managing larger workloads. Our design ensures efficient multi-node execution while leveraging remote shared storage for enhanced scalability. Experimental evaluations on an HPC cluster with up to 100 nodes demonstrate that SmartFlow achieves significantly higher throughput and lower latency than conventional consensus protocols. Future work will focus on 1) extending smart contract functionalities and enabling cross-chain interoperability to support a wider range of decentralized applications and 2) integrating SmartFlow with BAASH [2] to ex-

plore its scalability in extreme-scale workloads.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Krichen, M., Ammi, M., Mihoub, A. and Almutiq, M. (2022) Blockchain for Modern Applications: A Survey. *Sensors*, **22**, Article 5274. https://doi.org/10.3390/s22145274

[2] Mamun, A.A., Yan, F. and Zhao, D. (2021) BAASH: Lightweight, Efficient, and Reliable Blockchain-as-a-Service for HPC Systems. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, St. Louis, 14-19 November 2021, 1-18. https://doi.org/10.1145/3458817.3476155

[3] Al-Mamun, A., Li, T., Sadoghi, M., Jiang, L., Shen, H. and Zhao, D. (2019) Hpchain: An MPI-Based Blockchain Framework for Data Fidelity in High-Performance Computing Systems. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC*'19), Denver, 17-22 November 2019, 17-19.

[4] Al-Mamun, A., Yan, F. and Zhao, D. (2021) Scichain: Blockchain-Enabled Lightweight and Efficient Data Provenance for Reproducible Scientific Computing. 2021 *IEEE* 37*th International Conference on Data Engineering* (*ICDE*), Chania, 19-22 April 2021, 1853-1858. https://doi.org/10.1109/icde51399.2021.00166

[5] Al-Mamun, A., Li, T., Sadoghi, M. and Zhao, D. (2018) In-Memory Blockchain: Toward Efficient and Trustworthy Data Provenance for HPC Systems. 2018 *IEEE International Conference on Big Data* (*Big Data*), Seattle, 10-13 December 2018, 3808-3813. https://doi.org/10.1109/bigdata.2018.8621897

[6] Khan, S.N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E. and Bani-Hani, A. (2021) Blockchain Smart Contracts: Applications, Challenges, and Future Trends. *Peer-to-Peer Networking and Applications*, **14**, 2901-2925. https://doi.org/10.1007/s12083-021-01127-0

[7] Truby, J. (2018) Decarbonizing Bitcoin: Law and Policy Choices for Reducing the Energy Consumption of Blockchain Technologies and Digital Currencies. *Energy Research & Social Science*, **44**, 399-410. https://doi.org/10.1016/j.erss.2018.06.009

[8] Vukolić, M. (2016) The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In: *Open Problems in Network Security*: *IFIP WG* 11.4 *International Workshop*, Springer, 112-125.

[9] Dickerson, T., Gazzillo, P., Herlihy, M. and Koskinen, E. (2017) Adding Concurrency to Smart Contracts. *Proceedings of the ACM Symposium on Principles of Distributed Computing*, Washington, 25-27 July 2017, 303-312. https://doi.org/10.1145/3087801.3087835

[10] Dang, H., Dinh, T.T.A., Loghin, D., Chang, E., Lin, Q. and Ooi, B.C. (2019) Towards Scaling Blockchain Systems via Sharding. *Proceedings of the* 2019 *International Conference on Management of Data*, Amsterdam, 30 June-5 July 2019, 123-140. https://doi.org/10.1145/3299869.3319889

[11] Viriyasitavat, W., Xu, L.D., Niyato, D., Bi, Z. and Hoonsopon, D. (2022) Applications of Blockchain in Business Processes: A Comprehensive Review. *IEEE Access*, **10**, 118900-118925. https://doi.org/10.1109/access.2022.3217794

[12] Gu, W., Li, J. and Tang, Z. (2021) A Survey on Consensus Mechanisms for Blockchain Technology. 2021 *International Conference on Artificial Intelligence, Big Data and*

Algorithms (*CAIBDA*), Xi'an, 28-30 May 2021, 46-49. https://doi.org/10.1109/caibda53561.2021.00017

[13] Czech Financial Dataset (2024) Real Anonymized Transactions. https://data.world/lpetrocelli/czech-financial-dataset-real-anonymized-transactions

[14] Jin, C., Pang, S., Qi, X., Zhang, Z. and Zhou, A. (2021) A High Performance Concurrency Protocol for Smart Contracts of Permissioned Blockchain. *IEEE Transactions on Knowledge and Data Engineering*, **34**, 5070-5083. https://doi.org/10.1109/tkde.2021.3059959

[15] Foschini, L., Gavagna, A., Martuscelli, G. and Montanari, R. (2020) Hyperledger Fabric Blockchain: Chaincode Performance Analysis. 2020 *IEEE International Conference on Communications* (*ICC*), Dublin, 7-11 June 2020, 1-6. https://doi.org/10.1109/icc40277.2020.9149080

[16] Čeke, D., Kunosic, S. and Buzadija, N. (2022) Smart Contract Execution Costs Optimisation on Blockchain Network. 2022 45*th Jubilee International Convention on Information, Communication and Electronic Technology* (*MIPRO*), Opatija, 23-27 May 2022, 1442-1447. https://doi.org/10.23919/mipro55190.2022.9803805

[17] Wei, X., Sun, J., Qi, Z. and Fu, C. (2020) XuperChain: A Blockchain System That Supports Smart Contracts Parallelization. 2020 *IEEE International Conference on Smart Internet of Things* (*SmartIoT*), Beijing, 14-16 August 2020, 309-313. https://doi.org/10.1109/smartiot49966.2020.00055

[18] Liu, J., Li, P., Cheng, R., Asokan, N. and Song, D. (2021) Parallel and Asynchronous Smart Contract Execution. *IEEE Transactions on Parallel and Distributed Systems*, **33**, 1097-1108. https://doi.org/10.1109/tpds.2021.3095234

[19] Chung, M. and Park, C. (2022) Parallel Execution of Solidity Smart Contract Using Append-Only Shared Log. 2022 13*th International Conference on Information and Communication Technology Convergence* (*ICTC*), Jeju Island, 19-21 October 2022, 1722-1725. https://doi.org/10.1109/ictc55196.2022.9952656

[20] Fang, M., Zhou, X., Zhang, Z., Jin, C. and Zhou, A. (2022) SEFrame: An SGX-Enhanced Smart Contract Execution Framework for Permissioned Blockchain. 2022 *IEEE* 38*th International Conference on Data Engineering* (*ICDE*), Kuala Lumpur, 9-12 May 2022, 3166-3169. https://doi.org/10.1109/icde53745.2022.00289

[21] Tao, Y., Li, B., Jiang, J., Ng, H.C., Wang, C. and Li, B. (2020) On Sharding Open Blockchains with Smart Contracts. 2020 *IEEE* 36*th International Conference on Data Engineering* (*ICDE*), Dallas, 20-24 April 2020, 1357-1368. https://doi.org/10.1109/icde48307.2020.00121

[22] Li, C., Palanisamy, B. and Xu, R. (2019) Scalable and Privacy-Preserving Design of On/Off-Chain Smart Contracts. 2019 *IEEE* 35*th International Conference on Data Engineering Workshops* (*ICDEW*), Macao SAR, 8-12 April 2019, 7-12. https://doi.org/10.1109/icdew.2019.00-43

[23] Tan, S., S Bhowmick, S., Chua, H.E. and Xiao, X. (2020) LATTE: Visual Construction of Smart Contracts. *Proceedings of the* 2020 *ACM SIGMOD International Conference on Management of Data*, Portland, 14-19 June 2020, 2713-2716. https://doi.org/10.1145/3318464.3384687

[24] Jian, Z., Lu, Y., Qiao, Y., Fang, Y., Xie, X., Yang, D., *et al.* (2023) TSC-VEE: A Trustzone-Based Smart Contract Virtual Execution Environment. *IEEE Transactions on Parallel and Distributed Systems*, **34**, 1773-1788. https://doi.org/10.1109/tpds.2023.3263882

[25] Qi, X., Jiao, J. and Li, Y. (2023) Smart Contract Parallel Execution with Fine-Grained State Accesses. 2023 *IEEE* 43*rd International Conference on Distributed Computing*

Systems (*ICDCS*), Hong Kong SAR, 18-21 July 2023, 841-852. https://doi.org/10.1109/icdcs57875.2023.00068

[26] Guo, Y., Bland, W., Balaji, P. and Zhou, X. (2015). Fault Tolerant MapReduce-MPI for HPC Clusters. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Austin, 15-20 November 2015, 1-12. https://doi.org/10.1145/2807591.2807617

[27] Li, M., Hamidouche, K., Lu, X., Subramoni, H., Zhang, J. and Panda, D.K. (2016) Designing MPI Library with On-Demand Paging (ODP) of InfiniBand: Challenges and Benefits. *SC*16: I*nternational Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, 13-18 November 2016, 433-443. https://doi.org/10.1109/sc.2016.36

[28] AUHPC (2024) High Performance Computing Services. https://www.augusta.edu/it/hpcs/

[29] Zamani, M., Movahedi, M. and Raykova, M. (2018) RapidChain: Scaling Blockchain via Full Sharding. *Proceedings of the* 2018 *ACM SIGSAC Conference on Computer and Communications Security*, Toronto, 15-19 October 2018, 931-948. https://doi.org/10.1145/3243734.3243853

[30] Shifferaw, Y. and Lemma, S. (2021) Limitations of Proof of Stake Algorithm in Blockchain: A Review. *Zede Journal*, **39**, 81-95.

[31] Al-Mamun, A., Shen, H. and Zhao, D. (2022) DEAN: A Lightweight and Resource-Efficient Blockchain Protocol for Reliable Edge Computing. 2022 *IEEE International Parallel and Distributed Processing Symposium* (*IPDPS*), Lyon, 30 May-3 June 2022, 1261-1271. https://doi.org/10.1109/ipdps53621.2022.00125