Unleashing Multicore Strength for Efficient Execution of Blockchain Transactions

Ankit Ravish[1 [0009−0009−7355−8455]], Akshay Tejwani[1 [0009−0000−4732−8685]], Piduguralla Manaswini[1 [0000−0002−6295−0445]], and Sathya Peri[1 [0000−0002−3471−7929]]

Indian Institute of Technology Hyderabad, Telangana, India
{cs21resch11014,cs21mtech12015,cs20resch11007}@iith.ac.in, sathya_p@cse.iith.ac.in

Abstract. Blockchain technology is characterized by its distributed, decentralized, and immutable ledger system which serves as a fundamental platform for managing smart contract transactions (SCTs). However, these SCTs undergo sequential validation within a block which introduces performance bottlenecks in blockchain. In response, this paper introduces a framework called the Multi-Bin Parallel Scheduler (MBPS) designed for parallelizing blockchain smart contract transactions to leverage the capabilities of multicore systems. Our proposed framework facilitates concurrent execution of SCTs, enhancing performance by allowing non-conflicting transactions to be processed simultaneously while preserving deterministic order. The framework comprises of three vital stages: conflict detection, bin creation, and execution. We conducted an evaluation of our MBPS framework in Hyperledger Sawtooth v1.2.6, revealing substantial performance enhancements compared to existing parallel SCT execution frameworks across various smart contract applications. This research contributes to the ongoing optimization efforts in blockchain technology demonstrating its potential for scalability and efficiency in real-world scenarios.

Keywords: Blockchain · Smart Contracts · Parallel Execution · Conflict Detection.

1 Introduction

Blockchain [7] is a decentralized digital ledger offering secure and tamper-resistant record-keeping, with applications in finance, supply chain, and identity verification. Despite its advantages, blockchain scalability remains a challenge due to sequential smart contract execution. To address this, we propose the Multi-Bin Parallel Scheduler (MBPS) framework, enabling parallel smart contract execution to improve throughput and reduce transaction execution times.MBPS addresses the issue of transaction conflicts that can arise from parallel processing, ensuring consistent block validation across participants.

Numerous strategies have been explored to improve the efficiency of blockchain technology, particularly in terms of scalability, security, and consensus mechanisms. Sharding is a notable approach, dividing the blockchain network into smaller units or shards, which handle transactions independently to increase throughput by allowing concurrent transaction execution across shards. Dickerson et al. [5] proposed a speculative parallel execution model for miners and validators, utilizing software transactional memory to allow non-conflicting transactions to execute concurrently in a deterministic fork-join program. Similarly, Saraph et al. [10] introduced a concurrent execution model in Ethereum, dividing transactions into bins for parallel and serial processing based on their read and write sets. Optimistic Software Transactional Memory systems have also been leveraged for concurrent smart contract execution, demonstrating performance gains over sequential execution [2,3]. A direct acyclic graph (DAG)-based parallel scheduler has been proposed to enhance blockchain performance by improving the parallelism in smart contract execution [8]. Liu et al. [6] introduced an architecture where consensus nodes are separated from execution nodes, supporting asynchronous transaction ordering and parallel execution. The DiPETrans framework by Baheti et al. [4] enables distributed transaction execution with a collaborative PoW approach. Additionally, Yan et al. [11] presented an SCC-VS algorithm, optimizing concurrency within shards based on transaction characteristics like execution time and conflict rate. This paper presents three MBPS variations (Standard, Assisted, and Lockfree) and analyzes their effectiveness in improving throughput and preserving transaction order.

2 Proposed Framework

This section presents the design of our proposed Multi-Bin Parallel Scheduler (MBPS) framework, detailing its architecture and key components. The proposed MBPS framework facilitates parallel transaction execution while preserving a deterministic order, thereby leveraging the capabilities of multicore systems to enhance the efficiency of blockchain ecosystems. This framework introduces three distinct versions to address specific aspects of smart contract execution in blockchain ecosystems: Standard MBPS, Assisted MBPS, and Lockfree MBPS.

Each MBPS framework undergoes three crucial stages: *Conflict Detection, Bin Assignment, and Transaction Execution.* In **Conflict Detection**, conflicts between transactions T_i and T_j are identified if one transaction reads a data item that the other writes, or if both write to the same data item, as detailed in the paper [9]. During **Bin Assignment**, transactions are assigned to bins such that transactions within the same bin are independent, ensuring no conflicts within bins. The procedure for compact bin allocation is outlined in Algorithm 1. Finally, in **Transaction Execution**, non-conflicting transactions in each bin are executed sequentially, bin by bin, starting from Bin 1. All algorithms related to the frameworks discussed above are detailed in the paper [9].

The **Standard MBPS** framework employs synchronization barriers for parallel smart contract execution, comprising two phases: conflict set identification

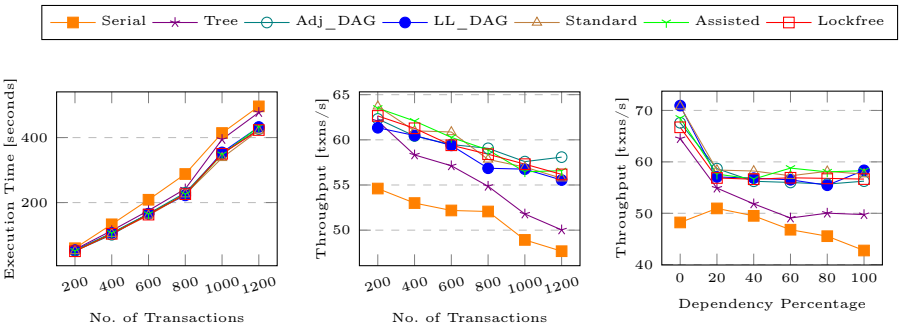
and bin number assignment. During conflict set identification, transactions are grouped based on conflicts (write-write, read-write, write-read) through input and output address analysis, allowing each transaction to be allocated to a conflict set in parallel. Subsequently, bin numbers are assigned to ensure that conflicting transactions are placed in separate bins, facilitating parallel execution. The **Assisted MBPS** enhances blockchain transaction execution by incorporating a barrier mechanism and helper threads to boost efficiency, particularly during thread crashes or latency. Similar to the Standard MBPS, it consists of conflict set identification and bin number assignment phases, augmented by helper threads to further optimize performance. In contrast, the **Lockfree MBPS** framework advances blockchain transaction parallelism by utilizing lock-free data structures, circumventing traditional synchronization methods like mutexes and barriers. This framework also includes the conflict set identification and bin number assignment phases, where helper threads and atomic operations create a lock-free environment. Algorithm 1 illustrates the bin assignment process using helper threads, enabling each thread to independently claim transactions, detect conflicts, and assign bins in parallel, thereby enhancing scalability and performance in blockchain execution.

3 Analysis of Experiments

In this section, we analyze the experiments conducted to evaluate our framework's performance with the Hyperledger Sawtooth blockchain [1]. We selected Sawtooth due to its parallelism support and inbuilt parallel scheduler. Although Sawtooth is Python-based, we implemented our multi-threaded MBPS framework in C++ for its low-level control over parallelism through threads, mutexes, and atomic operations. We conducted the experiments on an x86_64 machine with 56 CPUs, 2 threads per core, and 14 cores per socket (Intel Xeon CPU E5-2690 v4 @ 2.60GHz). Our MBPS framework's performance was compared with Sawtooth's parallel tree and serial schedulers, as well as the ADJ_DAG and LL_DAG frameworks [8]. Three experiment types were conducted: Baseline Performance Evaluation, Threads Latency Impact Analysis, and Threads Crash Resilience Analysis. The baseline performance evaluation compares execution times and throughput across frameworks, establishing standard metrics. The threads latency impact analysis explores how thread delays influence performance. Additionally, the threads crash resilience analysis focuses on performance under thread crashes, with particular attention to the lockfree bin scheduler, as it uniquely supports threads crash-handling compared to other schedulers. For these experiments, we used three conflict parameters (CP) defined in [8].

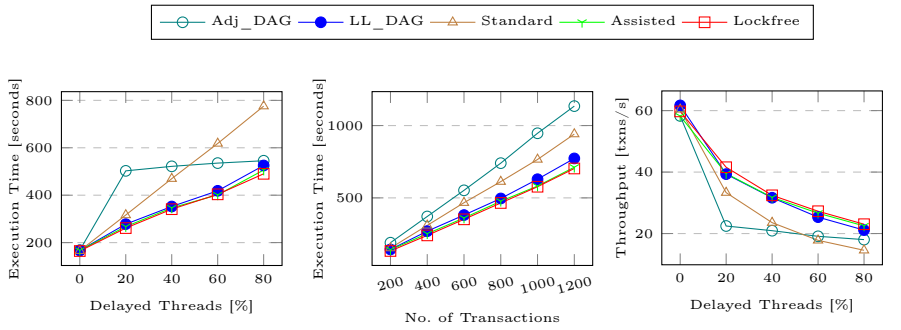
Algorithm 1 Bin Number Assignment Algorithm - Helper Threads

```
1: function BinNumAssignHelper
2:     processedTxns ← 0
3:     localCount ← 0
4:     flag ← 0
5:     while processedTxns < |Txns| do
6:         i ← atomicFetchAdd(i2, 1) mod |Txns|
7:         if initialBin[i] = −1 then
8:             localCount ← 0
9:             if flag = 1 then
10:                atomicFetchAdd(threadCounter2, −1)
11:            end if
12:            allotedBin ← CalculateBinHelper(i)
13:            if allotedBin = −1 then
14:                continue
15:            end if
16:            localVal ← allotedBin
17:            set<int> ∗ copy1, ∗copy2, ∗tempCopy
18:            repeat
19:                copy1 ← binArray[allotedBin]
20:                if copy1 = NULL then
21:                    (∗tempCopy).insert(i)
22:                    copy2 ← tempCopy
23:                else
24:                    if i ∈ ∗copy1 then
25:                        break
26:                    end if
27:                    for a ∈ ∗copy1 do
28:                        (∗copy2).insert(a)
29:                    end for
30:                    (∗copy2).insert(i)
31:                end if
32:            until binArray[allotedBin].CAS(copy1, copy2)
33:            temp1 ← −1
34:            if initialBin[i].CAS(temp1, localVal) then
35:                atomicFetchAdd(processedTxns, 1)
36:            end if
37:        else
38:            localCount ← localCount + 1
39:            if localCount = |Txns| and flag = 0 then
40:                flag ← 1
41:                atomicFetchAdd(threadCounter2, 1)
42:            end if
43:        end if
44:        if (threadCounter2 = numThreads) or (localCount = |Txns|) then
45:            atomicStore(processedTxns, |Txns|)
46:        end if
47:    end while
48: end function
```



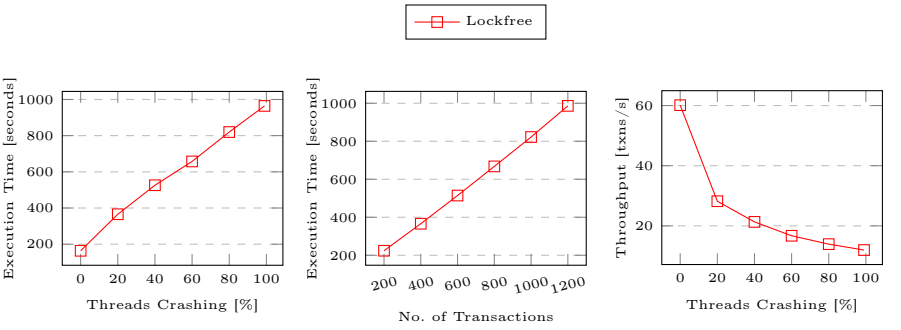
(a) CP1 (b) CP2 (c) CP3

Graph 1: Simple Wallet Smart Contracts - Baseline Performance Analysis



(a) 600 txns, CP1 (b) 33% delay ratio, CP2 (c) 600 txns, CP3

Graph 2: Simple Wallet Smart Contracts - Threads Latency Analysis



(a) 600 txns, CP1 (b) 33% crash ratio, CP2 (c) 600 txns, CP3

Graph 3: Simple Wallet Smart Contracts - Threads Crashing Analysis

4 Conclusion and Future Work

This paper introduces the MBPS framework for parallelizing blockchain smart contract transactions on multicore systems, featuring three variants: Standard, Assisted, and Lockfree MBPS. Each variant optimizes transaction execution while ensuring deterministic ordering. Experimental results on Hyperledger Sawtooth show notable improvements in throughput and execution time. Lockfree MBPS proved resilient to thread crashes, while Assisted MBPS effectively reduced latency via helper threads. Our future objective is to extend MBPS to distributed environments, addressing challenges like network latency, communication overhead, and synchronization. We also aim to optimize conflict detection and bin assignment to enhance efficiency.

References

1. Hyperledger Sawtooth Whitepaper. https://8112310.fs1.hubspotusercontent-na1.net/hubfs/8112310/Hyperledger/Offers/Hyperledger_Sawtooth_WhitePaper.pdf
2. Anjana, P.S., Attiya, H., Kumari, S., Peri, S., Somani, A.: Efficient concurrent execution of smart contracts in blockchains using object-based transactional memory. In: Networked Systems. pp. 77–93. Springer International Publishing, Cham (2021)
3. Anjana, P.S., Kumari, S., Peri, S., Rathor, S., Somani, A.: Optsmart: A space efficient optimistic concurrent execution of smart contracts. Distrib Parallel Databases (2022), https://link.springer.com/article/10.1007/s10619-022-07412-y
4. Baheti, S., Anjana, P.S., Peri, S., Simmhan, Y.: Dipetrans: A framework for distributed parallel execution of transactions of blocks in blockchain. Concurrency and Computation: Practice and Experience **n/a**, e6804 (2022). https://doi.org/https://doi.org/10.1002/cpe.6804, https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6804
5. Dickerson, T., Gazzillo, P., Herlihy, M., Koskinen, E.: Adding Concurrency to Smart Contracts. pp. 303–312. PODC '17, ACM, New York, NY, USA (2017)
6. Liu, J., Li, P., Cheng, R., Asokan, N., Song, D.: Parallel and asynchronous smart contract execution. IEEE Trans. Parallel Distrib. Syst. **33**(5), 1097–1108 (may 2022). https://doi.org/10.1109/TPDS.2021.3095234, https://doi.org/10.1109/TPDS.2021.3095234
7. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com (03 2009)
8. Piduguralla, M., Chakraborty, S., Anjana, P.S., Peri, S.: Dag-based efficient parallel scheduler for blockchains: Hyperledger sawtooth as a case study. In: Cano, J., Dikaiakos, M.D., Papadopoulos, G.A., Pericàs, M., Sakellariou, R. (eds.) Euro-Par 2023: Parallel Processing. pp. 184–198. Springer Nature Switzerland, Cham (2023)
9. Ravish, A., Tejwani, A., Manaswini, P., Peri, S.: Unleashing multicore strength for efficient execution of transactions (2024), https://arxiv.org/abs/2410.22460
10. Saraph, V., Herlihy, M.: An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts. Tokenomics '19 (2019)
11. Yan, W., Li, J., Liu, W., Tan, A.: Efficient concurrent execution of smart contracts in blockchain sharding. Security and Communication Networks **2021**, 1–15 (02 2021). https://doi.org/10.1155/2021/6688168