Poster: Enabling Cost-Effective Blockchain Applications via Workload-Adaptive Transaction Execution

A Case Study on Saving Fees for Write-intensive Accounts

Yibo Wang
Syracuse University
Syracuse, New York, USA
ywang349@syr.edu

Yuzhe Tang
Syracuse University
Syracuse, New York, USA
ytang100@syr.edu

ABSTRACT
As transaction fees skyrocket today, blockchains become increasingly expensive, hurting their adoption in broader applications.

This work tackles the saving of transaction fees for economic blockchain applications. The key insight is that other than the existing "default" mode to execute application logic fully on-chain, i.e., in smart contracts, and in fine granularity, i.e., user request per transaction, there are alternative execution modes with advantages in cost-effectiveness.

On Ethereum, we propose a holistic middleware platform supporting flexible and secure transaction executions, including off-chain states and batching of user requests. Furthermore, we propose control-plane schemes to adapt the execution mode to the current workload for optimal runtime cost.

We present a case study on the institutional accounts (e.g., coinbase.com) intensively sending Ether on Ethereum blockchains. By collecting real-life transactions, we construct workload benchmarks and show that our work saves 18% ~ 47% per invocation than the default baseline while introducing 1.81 ~ 16.59 blocks delay.

CCS CONCEPTS
• Security and privacy → Security protocols.

KEYWORDS
Blockchain, DApp, cost optimization, workload

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ACM Reference Format:
Yibo Wang and Yuzhe Tang. 2022. Poster: Enabling Cost-Effective Blockchain Applications via Workload-Adaptive Transaction Execution: A Case Study on Saving Fees for Write-intensive Accounts. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3548606.3563505

1 INTRODUCTION
Ethereum [1] is a blockchain platform that supports smart contracts. Ethereum charges a very high fee for transaction execution. One cause of the high fee is that data storage and processing are replicated to all nodes across a large network. Another cause is due to the basic principle of economy – raising user demand in transactions under the limited supply of block space (i.e., causing more users to bid for a single block slot). As a result, the Ether price has increased 27 times in the last two years (as of Mar. 2022). The high fees have real-world consequence: Ethereum clients are scared away and are forced to switch to other blockchains.

There are several cost saving schemes on blockchain that have been studied in the existing literature. iBatch [7] supports batching multiple smart-contract invocations into one transaction so that the base fee is amortized. GRuB [5] stores data off-chain in data-feed workloads and dynamically replicates the data on the blockchain. Layer-two protocols [2–4, 6] are built on top of the blockchain as extensions. They process the application logic off-chain to increase throughput and reduce gas fee. iBatch supports the invocations between any kinds of accounts, while is limited as it doesn’t support Ether transferring. GRuB is designed only for data-feed workloads and doesn’t support Ether transfer as well. Layer-two protocols support Ether transferring with the payment network. It saves Gas cost only for the transactions between institutional accounts.

While iBatch is under smart contract workloads and GRuB is under data feed workloads, it remains an open research problem: Can one optimize the cost of institutional client transferring Ether by leveraging batch and off-chain states?

We tackle this problem by presenting OCIA, a blockchain Optimization sCheme for Institutional Accounts via selectively placing states off the blockchain. OCIA is a control-plane scheme that maps the upper-layer transaction workloads (Ether transfers) to the underlying data plane that extends blockchains with batch transactions (i.e., iBatch) and off-chain states (i.e., GRuB). This work extends the workload to new, more common workloads, that is, Ether transfers from institutional accounts to average accounts.

We propose an (offline) dynamic optimization algorithm that decides when to upload the off-chain states to the on-chain storage (i.e., to convert an off-chain account to on-chain one) in order to maximize the batch size and minimize the cost.

We discover the institutional accounts who send the most transactions in one day Ethereum transaction history. We then analyzed the transaction history related with 3 institutional accounts as case study workloads including Coinbase, Ethermine, Crypto.com. The dynamic optimization algorithm makes one transaction contain more Ether transferring to amortize the cost while introducing a delay. We analyze the cost under the workload of write-intensive accounts with the dynamic optimization algorithm. The result shows OCIA saves 18% ~ 47% per invocation than baseline while has 1.81 ~ 16.59 blocks delay.


2 EXECUTION MODES AND COST

Preliminary: Ethereum transactions: An externally owned account (or EOA) in Ethereum is a public key whose secret key is only known by the user. An EOA is associated with the on-chain “state”, that is, the account balance. An Ethereum transaction is associated with the cost in the units of Gas. The transaction sender pays the transaction fee based on the Gas amount to miners to compensate their efforts in including the transaction. Each transaction has a base fee that is 21000 Gas covering the cost of verifying the signature of the sender as well as uploading the transaction data.

Blockchain transactions can be executed in different equivalent execution modes, including M1, M2, and M3, as shown in Figure 1.

2.1 Non-batched Tx and On-chain States (M1)

In Execution Mode M1, all accounts’ states are stored on the blockchain. To establish the cost model, we consider an over-simplified workload named W1’ as shown in Figure 3. In W1’, an institutional account A0 sends N Ether-transferring transactions in L blocks. The cost under the mode M1 is 21000 * N, where 21000 is the cost of sending an Ether-transferring transaction on Geth v1.10.17/Solidity 0.7.0. Thus, the per-transfer cost is C1 = 21000.

![Figure 1: Transaction execution modes.](image)

2.2 Batched Tx and On-chain States (M2)

In Execution Mode M2, accounts’ states are stored on the blockchain. The EOA’s request is batched with other requests, submitted at similar time, in a large transaction. As shown in Figure 1, the batch transaction is sent to a smart-contract, called Dispatcher, that dispatches the individual EOAs’ requests to their corresponding on-chain states. Specifically, the iBatch scheme [7] materializes M2 with an untrusted Batcher off-chain that is designed with the resilience against request-replay attacks.

Under the workload W1’(N, L), we use NW to denote the amount of Ether transfer of Tx1 in L blocks shown in Figure 3. The cost with M2 is 21000 * L + NW + [68 * (20 + 32) + 7500], where 68 + (20 + 32) is the transaction cost storing data field1 and 7500 is the cost of calling an internal Ether transfer. Thus, the per-transfer cost is C2 = 21000L/NW + 11036.

2.3 Batched Tx and Off-chain States (M3)

In Execution Mode M3, accounts’ states are stored off the blockchain and clients’ requests are batched in transactions.

Sender/receiver off-chain: The system is built on top of iBatch by placing the account states to an untrusted off-chain service. When a request, say Alice transferring Ether to Bob, is batched in a transaction, the on-chain smart contract needs to authenticate Alice’s and Bob’s old balances and update them, and compute and update the new digest including Alice’s and Bob’s updated balances. Thus, the transaction’s data includes the Merkle proofs authenticating Alice’s and Bob’s old balances against the old digest.

Sender on-chain, receiver off-chain: The M3 places the receivers’ states to an off-chain server and the state changes in a Hash log by expanding the Merkle tree instead of updating the state in off-chain directly. When receiving requests, say A0 transferring Ether to A2, we add dA1 and dA2 as new leaf nodes and keep the original balance unchanged. In each block, we upload the new digest and the state changes on-chain. The on-chain smart contract leverages the updating information and the old digest to authenticate and update the new digest. Under workload W1’, the cost is 21000 * L + NW + [68 * (20 + 36) + 222] + (68 * 32 + 5000) * L where 222 is the cost of SHA3 Keccak-256 operation of two 32 bytes digest, 68 * 32 is the cost of uploading the new digest in the call data field and 5000 is the cost of updating the digest for each block in the on-chain smart contract. Thus, the per-transfer cost is C3 = 28176L/NW + 4030.

3 COST ANALYSIS FOR WORKLOAD W1

The target workload pattern we optimize is shown in Figure 3. Given an institutional account A0, there are three transactions of interest in our workload: the transactions sent by A0, the transactions received by A1, and the transactions sent by A1.

M1’s costs under workload W1: For M1, the on-chain states of A1 are updated in Tx1. Hence, the cost of Tx2 is the same as a normal Ether transferring. Thus, the cost of W1 is C1 = 21000 * (NW + NR). Here, NW is the amount of Ether directly transferred from institutional account A0, and NR is the amount of Ether transferred within two hops from the institutional account (in Figure 3).

M2’s costs under workload W1: For M2, the on-chain states of A1 for ∀i ≤ 1 are updated in Tx1. Hence, the cost of Tx2 is the same as a normal Ether transferring. Thus, the cost of W1 is C2 = 21000 * L + 11036 + NW + 21000 * NR.

M3’s costs: When A1 for ∀i ≤ 1 under M3 sends Ether to another account, the off-chain server push all the A1 related states on-chain, includes the balance state and the updating value in every blocks. The off-chain server also uploads the Merkle proofs of each related account’s off-chain state. The on-chain smart contract leverages the Merkle proofs to authenticate the updated balance of A1. Under workload W1’, say A1 with an initial balance in off-chain server, it


---

168 is Gas cost per byte in the data field and the 20 and 32 respectively refer to the number of bytes storing the receiver address and value in an Ether transfer.
We download the Ether transferring transaction history and find which cost.

The amortized cost per transaction of $M_1$ is $21000$, which saves $61\%$ than baseline. The amortized cost of $M_2$ is $18341$, which saves $12.66\%$ than baseline. The amortized cost of $M_3$ is $42143$, which costs $100.68\%$ more than baseline.

**Ethermine**: Ethermine is a big miner in Ethereum. The average amount of $NW$ and $NR$ per block are $12.30$ and $25.62$.

$W_1$, the amortized cost of $M_1$ is $21000$. The amortized cost of $M_2$ is $18341$, which saves $12.66\%$ than baseline. The amortized cost of $M_3$ is $42143$, which costs $100.68\%$ more than baseline.

**Crypto.com**: Crypto.com is also a cryptocurrency exchange platform. The average amount per block of $NW$ and $NR$ are $4.65$ and $4.88$. The amortized cost of $M_2$ and $M_3$ are $21497$ and $66698$, which cost $2.37\%$ and $217.61\%$ more than baseline.

**Cost Evaluation under Real Workloads**

The amortized cost per transaction of $M_3$ depends on the number of $NW$ and $NR$. We find the accounts that meet the conditions $C_3 = 21000 < 0$ as write-intensive accounts, that is the accounts whose $NW$ and $NR$ subject to $NW < (8485+NW−14088)/25843$ and $NW > 1.66$. We get the transaction history of the write-intensive accounts as write-intensive workloads. 13.00% of accounts in Ethermine workloads are write-intensive accounts. The workloads of Coinbase and Crypto.com have 2.14% and 0.56% write-intensive accounts.

We dynamically upload the off-chain states to the on-chain storage when the $NW$ and $NR$ in the off-chain state meet the write-intensive condition. With this policy, the cost of each transaction is less than baseline while it introduces delay. We analyzed the average cost of each transaction and the average delay of the write-intensive workloads in table 1.

The above optimizing cost policy saves cost while the delay is relatively large. We propose various delay policies to make trade-off between cost saving and delay. Max_0 policy: updates the on-chain state with no delay. It updates the on-chain state in the same block interval with the Ether transferring requests. There is no delay introduced in Max_0 policy, while it hardly achieves a positive cost saving because few requests are batched in the transaction. Max_5 policy: updates the on-chain state with at most 5 blocks delay. If the $NW$ and $NR$ meet the conditions of saving cost within 5 blocks, we send a transaction to update the on-chain state. Otherwise, we update the on-chain state no matter what the $NW$ and $NR$ is to make sure the delay is not larger than 5. In the same way, we have Max_10 policy and Max_15 policy. The trade-off of cost saving and delay with various delay policies is illustrated in figure 4.

| Table 1: Cost and delay of write-intensive workloads |
|-----------------------------------------------|
| Normalized cost | Coinbase | Ethermine | Crypto.com |
|-----------------|----------|-----------|------------|
| Average delay (block) | 1.81 | 5.24 | 16.59 |

5 ACKNOWLEDGEMENT

The authors are partially supported by the National Science Foundation under Grants CNS2139801, CNS1815814 and DGE2104532.

REFERENCES

[1] Retrieved June, 2021. Ethereum project. https://www.ethereum.org/
[2] Retrieved June, 2021. Lightning network. Scalable, Instant Bitcoin/Blockchain Transactions. https://lightning.network
[3] Raymond Cheng, Fan Zhang, Jermei Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2018. Ekeden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. CoRR abs/1804.05141 (2018). arXiv:1804.05141 http://arxiv.org/abs/1804.05141
[4] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General State Channel Networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. David Lie, Mohammad Mannan, Michael Backes, and Xiaofei Wang (Eds.). ACM, 949–966. https://doi.org/10.1145/3243734.3243856
[5] Kai Li, Yuzhe Richard Tang, Jiaqi Chen, Zhehu Yuan, Cheng Xu, and Jianliang Xu. 2020. Cost-Effective Data Feeds to Blockchains via Workload-Adaptive Data Replication. In Middleware ’20, 21st International Middleware Conference, Delft, The Netherlands, December 7-11, 2020, Ulima Da Silva and Radiger Kapitza (Eds.). ACM, 371–385. https://doi.org/10.1145/3423111.3423569
[6] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. 2017. Sprites: Payment Channels that Go Faster than Lightning. CoRR abs/1702.05082 (2017). arXiv:1702.05082 http://arxiv.org/abs/1702.05082
[7] Yibo Wang, Qi Zhang, Kai Li, Yuzhe Tang, Xiupei Luo, and Ting Chen. 2021. iBatch: Saving Ethereum Fees via Secure and Cost-Effective Batching of Smart-Contract Invocations. CoRR abs/2106.08554 (2021). arXiv:2106.08554 https://arxiv.org/abs/2106.08554