Parallelized Ouroboros Praos

ALEX SIERKOV

Proof-of-stake blockchains, such as Cardano, solve the energy consumption problem of proof-of-work ones and make the cost of 51% attacks prohibitive. However, they introduce a new scalability problem for wallet nodes—an increased cost of block verification because determining the stake of block-signing nodes requires tracking most blockchain updates. In Cardano, due to the unsatisfactory performance of its only full data wallet, Daedalus, many users are forced to use less secure light wallets even when they do not want to. This paper presents an open-source parallelized implementation of Cardano's consensus algorithm, Ouroboros Praos, optimized for the batch tracking of blockchain updates. The method is compared against the available alternatives, Cardano Node and Mithril, and achieves state-of-the-art performance in both full snapshot and incremental synchronization cases for Cardano. The method can be a relevant comparison point and source of optimization ideas for other proof-of-stake blockchains.

CCS Concepts: • **Theory of computation → Cryptographic protocols**; **Parallel algorithms**; • **Information systems → Open source software**.

Additional Key Words and Phrases: Cardano, blockchain synchronization, consensus algorithms, batch processing

1 INTRODUCTION

Effective use of parallel processing is a critical lever for software projects that must satisfy the performance expectations of their users while facing an exponential data growth. Public blockchains are good examples of such projects. However, Cardano, the eighth public blockchain based on market capitalization as of February, 2024,[1], has long been underutilizing this lever, explaining that position[2] as being due to the sequential nature of its consensus algorithm, Ouroboros Praos [18]. This paper presents an open-source implementation [7, 10] of a parallelized version of Ouroboros Praos that is tailored for the use case of personal wallets, blockchain clients that track updates only sporadically, and, thus, suffers the most from low data validation performance.

Ouroboros Praos is a proof-of-stake consensus algorithm in which the eligibility of a node to generate the next block depends on the amount of assets delegated to it by the chain participants. Furthermore, the participants are rewarded for their participation in block generation. Thus, the amount of assets influencing block production keeps changing, even if all participants abstain from transactions. For this reason, it was believed to be impossible to noticeably increase the data processing throughput of Ouroboros Praos through parallel processing. This position was unfavorable to Cardano's scalability because the single-core performance of modern processors has been growing much slower than their parallel-processing performance.

To solve the problem of slow bootstrapping of new nodes, a sampling-based multi-signature scheme, Mithril [9, 16], was introduced to the Cardano ecosystem. Mithril represents a group of solutions, such as PoPoS [13] or FlyClient [15], that reduce verification complexity through "compressing" techniques, such as sampling and Merkle trees. However, Mithril comes with its own limitations. It requires the majority of block-producing nodes to operate additional software for signing blockchain snapshots, and it does not currently support incremental snapshots, despite partial synchronization being the most frequent use case for personal wallets.

While looking for pragmatic solutions, it was difficult not to notice the improvements in the parallel-processing capabilities of modern consumer-grade processors, leading to a research question: Is it possible to optimize the execution of Ouroboros Praos to such an extent that a high-end

[1]According to coinmarketcap.com [11]
[2]Charles Hoskinson, the founder of Cardano, shared this position in a video [20] in which he refutes the benefits of parallel processing to accelerate the synchronization performance of Cardano's Daedalus wallet.

Internet connection becomes the bottleneck of the synchronization rather than the computational power? A positive answer to this question would neutralize the key benefits of the compressing techniques while benefitting from the ability to reuse the existing block-signing infrastructure.

The contributions of this paper are as follows:

- Showing that it is possible to achieve high levels of parallel efficiency despite the significant number of sequential steps present in the original Ouroboros Praos algorithm;
- Achieving state-of-the-art performance in full snapshot synchronization: in the 24-core configuration, the method is 210 times quicker than Cardano Node and 3.7 times quicker than Mithril;
- Achieving state-of-the-art performance in incremental synchronization: in the 8-core and 24-core configurations, the method is more than 10 times quicker than both Cardano Node and Mithril for synchronization intervals between six months and one week of data;
- Providing evidence that the data-processing throughput of Cardano Node, the only complete implementation of Cardano protocols, can noticeably benefit from further optimization, such as a deeper integration of parallel processing methods.

2 OVERVIEW OF OUROBOROS PRAOS

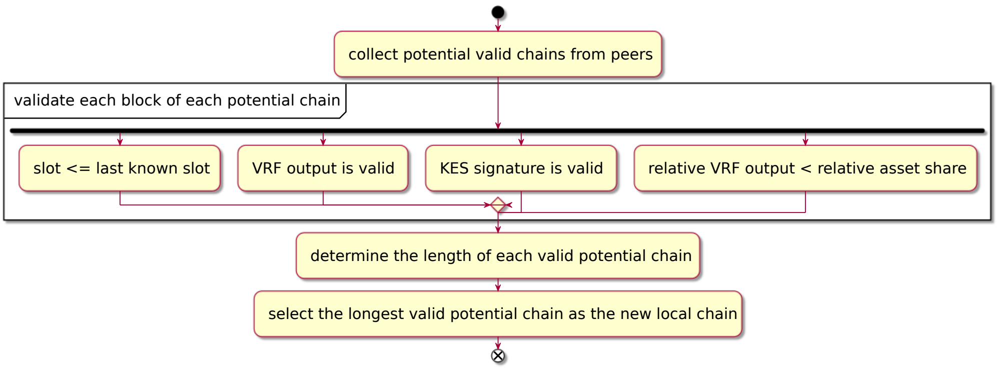

collect potential valid chains from peers
validate each block of each potential chain
slot <= last known slot
VRF output is valid
KES signature is valid
relative VRF output < relative asset share
determine the length of each valid potential chain
select the longest valid potential chain as the new local chain

Fig. 1. Ouroboros Praos's chain update rule

Ouroboros Praos[18] is a proof-of-stake consensus algorithm introduced in 2017. It has been supporting Cardano's mainnet since 2020. It is designed to work under the honest majority assumption that the majority of assets, "stake" in the protocol's terms, is delegated to honest nodes, the ones that precisely follow the protocol's specification.

Figure 1 presents a schematic overview of Ouroboros Praos's chain update rule. This rule is responsible for processing all incoming blockchain updates from other nodes and represents the main computation loop of a wallet node when it is synchronizing its local state. At first glance, several potential opportunities for parallelization are present:

- Parallelize the validation of alternative potential chains
- Parallelize the validation of individual blocks within the same potential chain
- Parallelize the validation of individual steps of a block's validation

However, several nuances make the task more complex in practice:

- The knowledge of the last-known slot[3] needs to be shared across threads performing block validation of the same potential chain.
- The validation of outputs of verifiable random functions [19] (VRFs) requires the knowledge of a nonce, a unique value computed from data of blocks in the preceding epoch[4] and the nonce of the previous epoch as well.
- The validation of key evolving signatures,[5] KES, requires tracking of the preceding block signatures by the same signer.
- The computation of the relative stake of a block signer requires the processing of virtually all blockchain updates—the key reason why parallelizing Ouroboros Praos was believed to have low optimization potential.

To further highlight the complexity of parallelizing the tracking of blockchain updates, here are the state-tracking activities that need to be considered:

- Registrations, retirements, and parameter updates of stake pools, the block-signing entities in the Cardano blockchain
- Registrations, retirements, and delegations of assets of individual accounts
- Transactions affecting assets attributable to each individual account
- Distribution of staking rewards as a by-product of block generation
- Changes to the protocol parameters

3 OVERVIEW OF THE PROPOSED METHOD

Figure 2 presents an overview of the parallelized validation of a single potential chain. That is the most complex part of the method, and it can be extended to the validation of multiple potential chains by applying a trivial parallel loop over the available potential chains.

The main idea behind the method is that to achieve high levels of parallel efficiency on high-end consumer hardware, it is sufficient for the sequential path of the algorithm to be fast enough to not starve the parallel execution of CPU-heavy computations, such as cryptographic operations. Therefore, to reduce the time spent in the sequential path, several adjustments are applied to the original Ouroboros Praos:

- Input data are pre-aggregated in parallel where possible.
- The execution of computations is delayed until the point when they are needed by the sequential path.
- The execution of CPU-heavy computations is transferred to a background low-priority thread pool.
- The data structures are reviewed to reduce the time spent in the sequential path.
- On-disk data structures are used to reduce inter-thread locking.
- ZStandard [17] data compression is used in on-disk data structures to reduce the consumed I/O bandwidth and storage space.
- C++ programming language is used for the implementation to have control over the memory layout of data structures and to benefit from machine-specific optimizations of the compiled code.

The following section describes key individual components of the method.

[3]Time segments of one-second duration, which are used for leader election in Cardano

[4]A Cardano blockchain segment corresponding to 432,000 seconds of real-world time.

[5]Presented in Appendix A.3 of the Ouroboros Praos paper [18]

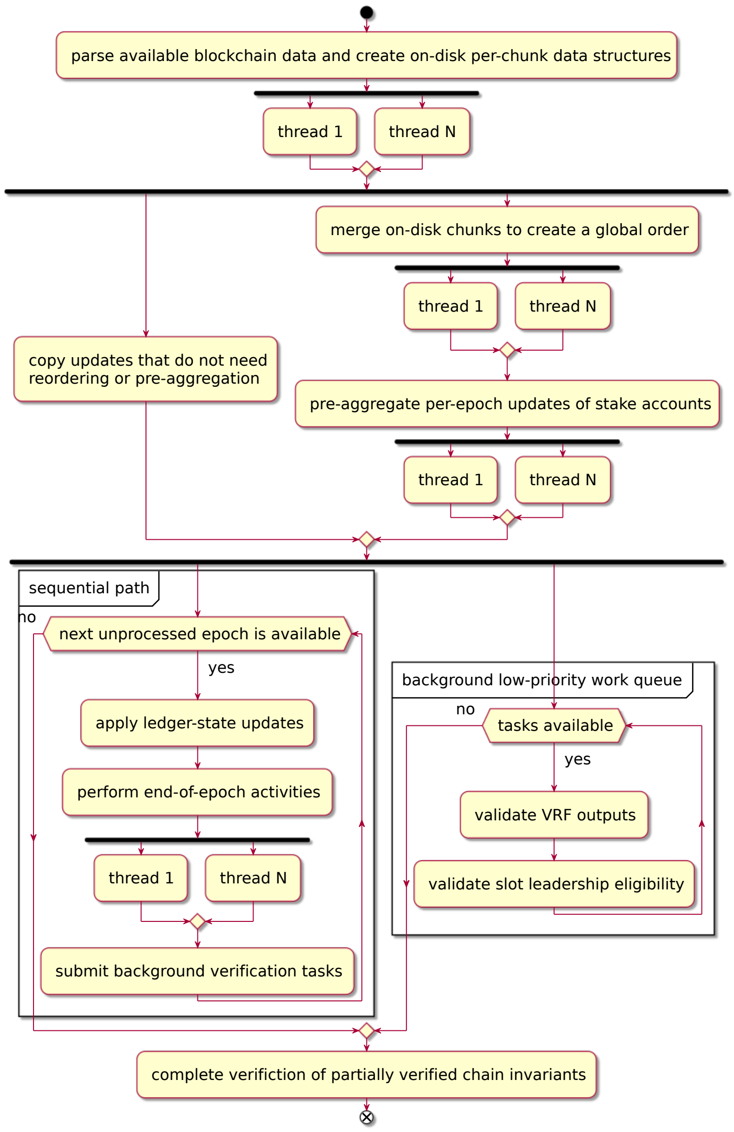

parse available blockchain data and create on-disk per-chunk data structures
thread 1
thread N
merge on-disk chunks to create a global order
thread 1
thread N
copy updates that do not need reordering or pre-aggregation
pre-aggregate per-epoch updates of stake accounts
thread 1
thread N
sequential path
no
next unprocessed epoch is available
yes
apply ledger-state updates
perform end-of-epoch activities
thread 1
thread N
submit background verification tasks
background low-priority work queue
no
tasks available
yes
validate VRF outputs
validate slot leadership eligibility
complete verifiction of partially verified chain invariants

Fig. 2. Overview of the parallelized validation of an individual potential chain

4 DESCRIPTION OF INDIVIDUAL COMPONENTS

4.1 Parallel parsing and creation of supporting on-disk data

The initial data preparation is based on the ideas presented in [21]. The key aspects of the preparation are as follows:

- Parse blockchain data in parallel, with each thread creating chunks of several task-related indices.
- Use a hybrid radix/merge sort algorithm leveraging the uniform distribution of index keys representing Blake2b hashes[14].

- Reference individual blockchain entries by their absolute byte offsets.

The data structures of individual indices are listed in Appendix B. Briefly, they contain the following information:

- Individual transaction outputs, their amounts, and referenced stake IDs
- References to used transaction outputs along with the epoch in which they were used
- VRF data of individual blocks
- Aggregated counts of blocks generated by stake pools for each epoch
- Sums of fees
- Time-based updates, such as pool and stake registrations and deregistrations, stake delegations, reward withdrawals and instant transfers, and protocol-parameter updates

Furthermore, to make the on-disk data easier to process in parallel and reduce the required disk I/O, indices are normally partitioned by the first byte of their respective entries and compressed using the ZStandard [17] compression algorithm.

4.2 Parallel join of hash-based indices

Cardano uses an extended unspent transaction outputs transaction model, in which the spent assets of a transaction are formed from the outputs of a previously conducted transaction. For this reason, to track asset outflows of an account, one needs to track new transactions that reference transaction outputs (TXO) referencing that account.

The proposed method handles this by creating two on-disk indices:

- TXO index—records the amount of assets tied to a given transaction output
- TXO-use index—records the use of transaction outputs as inputs in new transactions

To prepare per-account information about asset outflows stored in Outflows index, the method joins the two above indices in the following way:

- Each index is partitioned by the first byte of the respective transaction hash value.
- The join operation is performed in parallel with same-byte-prefixed partitions of both indices joined by a dedicated thread.
- The joined data are kept in the partitioned form to support parallelized search operations.

4.3 Parallel pre-aggregation of stake account updates

The pre-aggregation of stake account updates is necessary for quicker transitions of the ledger state. This is done by joining Inflows and Outflows indices[6] by stake-account IDs. The resulting data are additionally partitioned by epoch with all individual account updates (positive amounts for inflows and negative amounts for outflows) aggregated into a single signed integer—per-account per-epoch balance change. Section 4.2.3 of [21] explains this method in more detail.

4.4 Parallel computation of staking rewards

For every epoch, once the necessary data, such as the epoch's reward pot, per-account assets and stake delegation information, are available, the parallel computation of staking rewards begins. First, per-pool reward pots and bases for member reward computations are prepared. After that, all available stake accounts are partitioned by the first byte of their stake IDs, and rewards for each partition are computed in parallel.

[6]Details of the index structure are presented in Appendix B.

4.5 Parallelized end-of-epoch ledger-state transition

At the end of each epoch, snapshots of certain ledger dictionaries are created for future staking reward calculations, such as snapshots of stake balances and pool delegations. These dictionaries contain millions of entries, and depending on the chosen data structure, copying them may be costly.

In addition to selecting more optimal data structures, it is possible to parallelize the snapshot creation by copying each dictionary in parallel and partitioning dictionaries by the first byte of the stake IDs when further parallelization is beneficial.

4.6 Parallel verification of slot leadership eligibility

Once an epoch's VRF nonce and the corresponding pool stake distribution are prepared by the sequential path of the algorithm, slot leadership eligibility verification tasks are added to the low-priority background work queue. Each task executes a small batch[7] of verifications and does the following:

- Checks the validity of the VRF's output for the block's slot
- Checks that the VRF's output is smaller than the threshold computed from the relative share of the signing node's stake

4.7 Parallel verification of KES signatures

The cryptographic validity of KES signatures is checked during the initial parsing of blockchain data. However, since each chunk is parsed independently and in parallel, the information about the parameters of previously used KES signatures by the same issuer is not available at that time. Therefore, to evaluate the validity of sequential invariants of KES signatures, the thread performing the initial validation of a blockchain chunk validates that within that chunk the invariants hold. After that, it creates an in-memory data structure for final processing. This data structure contains the earliest and the latest KES counters seen for each signing node in that blockchain chunk. The overall validity of all KES signatures is checked at the end of the computation when the data structures from all threads are available and can be globally ordered.

4.8 Sequential path

Figure 2 in a block "sequential path" highlights the steps of the algorithm that need to be taken in order. These are the applications of pre-aggregated updates to transition the ledger state from one epoch to another. The arising verification tasks are submitted to a background low-priority work queue once there is sufficient data for them.

4.9 Final verification of partially verified chain invariants

Several chain invariants can only be partially verified during the initial parsing. Therefore, a final verification pass is done once all chunks are parsed and a global order of the necessary items can be established. These final verifications are:

- Verification of the monotonicity of KES signatures
- Verification of the monotonicity of block slot numbers
- Verification of the coherency of the chain of block header hashes

5 RESULTS

To evaluate the method, a publicly rentable 24-core server was used. The server's core count was selected at the limit of what is possible with the most powerful laptop CPUs, such as Intel's 14900HX

[7] 250 currently

[8] with 24 cores and Apple's M3 Max [1] with 16 cores. All experiments were repeated three times, and the average value is reported. The exact configuration of the benchmarking environment and the links to the source code of the benchmarks are presented in Appendix C.

5.1 Data-processing throughput and parallel efficiency

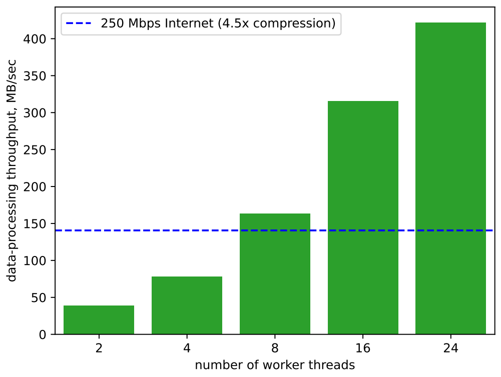


Fig. 3. Data-processing throughput

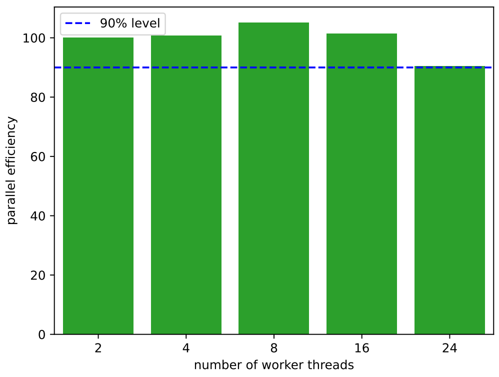


Fig. 4. Parallel efficiency

To evaluate the performance of the proposed approach, the method's run time was measured with two, four, eight, 16, and 24 worker threads[8]. The measured run times were used to compute the data-processing throughput as the validated blockchain size divided by the run time. Figure 3 presents the results. In all configurations with eight or more threads, the method can fully saturate a 250 Mbps Internet connection transferring compressed blockchain data,[9] thus, making the speed of the Internet connection the bottleneck in most consumer configurations.

To evaluate the performance scaling, the parallel efficiency coefficient was computed for all tested thread counts as the throughput for that worker count divided by the estimated linear scaling of the two-thread configuration[10]. Figure 4 presents the results, which show that the method reaches a 90%+ parallel efficiency level for all evaluated worker counts.

5.2 End-to-end synchronization time

To evaluate the end-to-end synchronization time, a compressed copy of the Cardano blockchain data was uploaded to a dedicated server with 1 Gbps outbound Internet connection and shared over HTTP protocol. Then, a version of the code that simultaneously downloads and processes the data was run utilizing either 24 cores (representing the best-case scenario) or eight cores (representing a typical consumer laptop).

Figure 5 presents the results and compares them to the results of Cardano Node [6], a status quo implementation of Cardano specifications, and Mithril [9], a Cardano custom solution for the delivery of blockchain snapshots. In all measured cases, the proposed method synchronized the blockchain faster than the alternatives, therefore, achieving state-of-the-art results for the case of

[8]The configuration with one worker was not evaluated because the current implementation requires a minimum of two workers.

[9]A 4.5x compression ratio is achieved by the Zstandard [17] algorithm at its level 22.

[10]The two-thread throughput was divided by two and multiplied by the thread count for which the parallel efficiency coefficient was computed

complete synchronization. In the 24-core configuration, the method was 210 times quicker than Cardano Node and 3.7 times quicker than Mithril.

Figure 6 additionally presents the computed end-to-end data-processing throughput to compare the impact of adding the download step on the method's performance.

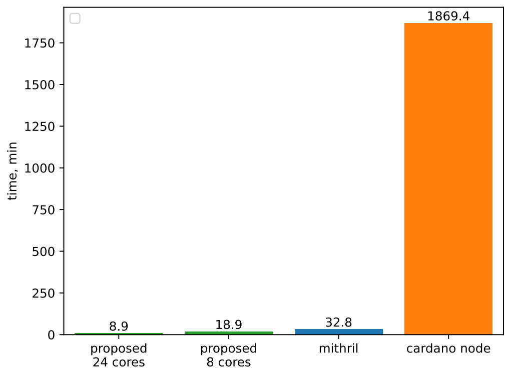


Fig. 5. End-to-end synchronization time

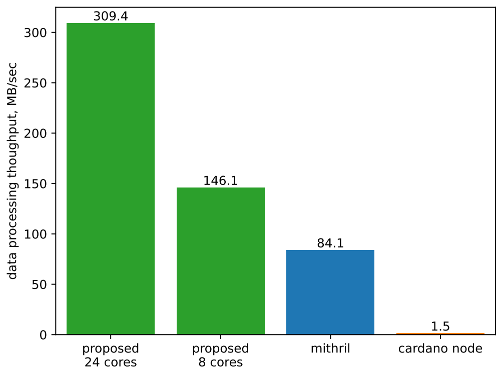


Fig. 6. End-to-end throughput

5.3 Incremental synchronization time

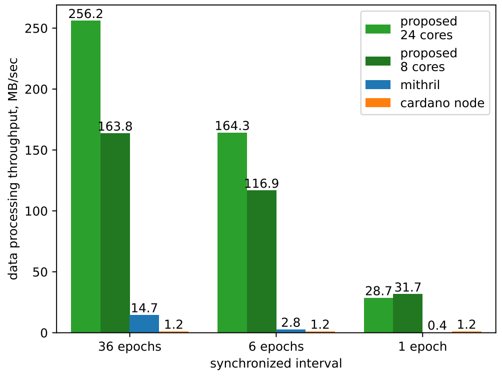


Fig. 7. Incremental throughput

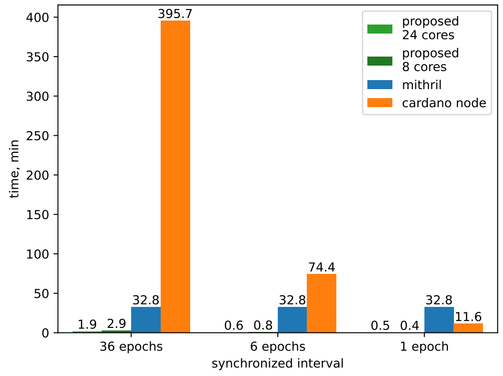


Fig. 8. Incremental synchronization time

Finally, the method was evaluated in the "partial synchronization" scenario typical for personal wallets, where wallet software is synchronized only sporadically. Three synchronization intervals were chosen to reflect a diverse range of potential use cases ranging between weekly use and the recovery of a forgotten wallet:

- 36 Cardano epochs, representing a six-month pause.
- six Cardano epochs, representing a 1-month pause.
- one Cardano epoch, representing a five-day pause.

The method was again compared against Cardano Node and Mithril. It is important to note that Mithril currently does not support incremental synchronization, and its results are presented for completeness and represent the time of a full snapshot delivery.

Figure 7 presents the data-processing throughput, and Figure 8 presents the incremental synchronization time. In all evaluated cases, the performance of the proposed method is more than 10 times better than the alternatives, thus, achieving state-of-the-art results for the case of incremental synchronization as well.

6 CONCLUSION

This paper presents an open-source parallelized implementation of Cardano's consensus algorithm, Ouroboros Praos. Despite the previously widespread beliefs that Ouroboros Praos cannot noticeably benefit from parallelization, the method achieves 90%+ levels of parallel efficiency in the range of core counts relevant for high-end consumer hardware: between two and 24. With this result, the method achieves state-of-the-art performance in full snapshot synchronization. In the 24-core configuration, it is 210 times quicker than Cardano Node and 3.7 times quicker than Mithril.

Furthermore, the method achieves state-of-the-art performance in incremental synchronization as well. In both eight-core and 24-core configurations, the method is more than 10 times quicker than both Cardano Node and Mithril when the synchronized interval is between six months and one week of data.

Finally, this paper presents a list of practical optimization ideas that can be used to improve the performance of Cardano Node, Cardano's software for block-producing nodes, and, thus, to have a strong positive influence on the scalability of the Cardano blockchain overall.

REFERENCES

[1] 2024. Apple MacBook Pro specifications. ttps://apple.com/macbook-pro/specs/

[2] 2024. Benchmark scripts for the Cardano Node's synchronization performance in the Daedalus Turbo GitHub repository. https://github.com/sierkov/daedalus-turbo/tree/parallelized-ouroboros-praos/experiment/sync-cardano-node

[3] 2024. Benchmark scripts for the Mithril's synchronization performance in the Daedalus Turbo GitHub repository. https://github.com/sierkov/daedalus-turbo/tree/parallelized-ouroboros-praos/experiment/sync-mithril

[4] 2024. Benchmark scripts for the proposed method's end-to-end synchronization performance in the Daedalus Turbo GitHub repository. https://github.com/sierkov/daedalus-turbo/tree/parallelized-ouroboros-praos/experiment/sync-full

[5] 2024. Benchmark scripts for the proposed method's incremental synchronization performance in the Daedalus Turbo GitHub repository. https://github.com/sierkov/daedalus-turbo/tree/parallelized-ouroboros-praos/experiment/sync-incremental

[6] 2024. Cardano Node Github repository. https://github.com/IntersectMBO/cardano-node

[7] 2024. Daedalus Turbo GitHub repository. https://github.com/sierkov/daedalus-turbo

[8] 2024. Intel Core i9 processor 14900HX specifications. https://www.intel.com/content/www/us/en/products/sku/235995/intel-core-i9-processor-14900hx-36m-cache-up-to-5-80-ghz/specifications.html

[9] 2024. Mithril GitHub repository. https://github.com/input-output-hk/mithril

[10] 2024. Parallelized Ouroboros Praos branch in the Daedalus Turbo GitHub repository. https://github.com/sierkov/daedalus-turbo/tree/parallelized-ouroboros-praos

[11] 2024. Today's cryptocurrency prices by market cap. https://coinmarketcap.com/

[12] 2024. Vultr cloud hosting and services. https://www.vultr.com/

[13] Shresth Agrawal, Joachim Neu, Ertem Nusret Tas, and Dionysis Zindros. 2023. Proofs of Proof-Of-Stake with Sublinear Complexity. In *5th Conference on Advances in Financial Technologies, AFT 2023, October 23-25, 2023, Princeton, NJ, USA (LIPIcs, Vol. 282)*, Joseph Bonneau and S. Matthew Weinberg (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:24. https://doi.org/10.4230/LIPICS.AFT.2023.14

[14] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: Simpler, Smaller, Fast as MD5. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7954)*, Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini (Eds.). Springer, 119–135. https:

//doi.org/10.1007/978-3-642-38980-1_8

[15] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. 2020. FlyClient: Super-Light Clients for Cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020.* IEEE, 928–946. https://doi.org/10.1109/SP40000.2020.00049

[16] Pyrros Chaidos and Aggelos Kiayias. 2021. Mithril: Stake-based Threshold Multisignatures. *IACR Cryptol. ePrint Arch.* (2021), 916. https://eprint.iacr.org/2021/916

[17] Yann Collet. 2021. Zstandard Compression and the 'application/zstd' Media Type. (2021). https://datatracker.ietf.org/doc/html/rfc8878

[18] Bernardo Machado David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2017. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. *IACR Cryptol. ePrint Arch.* (2017), 573. http://eprint.iacr.org/2017/573

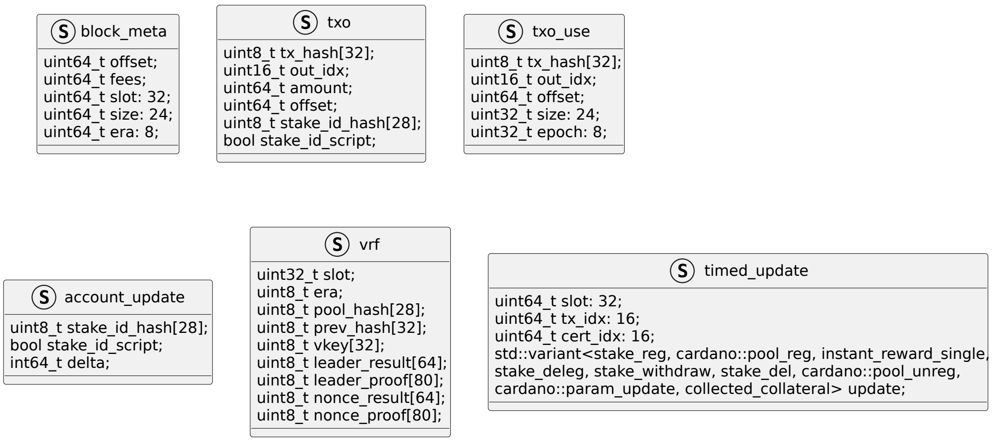
[19] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function with Short Proofs and Keys. In *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3386)*, Serge Vaudenay (Ed.). Springer, 416–431. https://doi.org/10.1007/978-3-540-30580-4_28

[20] Charles Hoskinson. 2022. Comments on Daedalus Turbo project. https://www.youtube.com/watch?v=z9uM2AVGU-g

[21] Alex Sierkov. 2023. Highly Parallel Reconstruction of Wallet History in the Cardano Blockchain. (2023). https://github.com/sierkov/daedalus-turbo/blob/main/doc/2023_Sierkov_WalletHistoryReconstruction.pdf

A APPENDICES

B ON-DISK DATA STRUCTURES



```
S block_meta
uint64_t offset;
uint64_t fees;
uint64_t slot: 32;
uint64_t size: 24;
uint64_t era: 8;
```

```
S txo
uint8_t tx_hash[32];
uint16_t out_idx;
uint64_t amount;
uint64_t offset;
uint8_t stake_id_hash[28];
bool stake_id_script;
```

```
S txo_use
uint8_t tx_hash[32];
uint16_t out_idx;
uint64_t offset;
uint32_t size: 24;
uint32_t epoch: 8;
```

```
S account_update
uint8_t stake_id_hash[28];
bool stake_id_script;
int64_t delta;
```

```
S vrf
uint32_t slot;
uint8_t era;
uint8_t pool_hash[28];
uint8_t prev_hash[32];
uint8_t vkey[32];
uint8_t leader_result[64];
uint8_t leader_proof[80];
uint8_t nonce_result[64];
uint8_t nonce_proof[80];
```

```
S timed_update
uint64_t slot: 32;
uint64_t tx_idx: 16;
uint64_t cert_idx: 16;
std::variant<stake_reg, cardano::pool_reg, instant_reward_single,
stake_deleg, stake_withdraw, stake_del, cardano::pool_unreg,
cardano::param_update, collected_collateral> update;
```

Fig. 9. C-style definitions of on-disk data structures

The format of on-disk data structures is based on several simple principles:

- Use fixed-size elements to make on-disk and in-memory access of an item by its index an O(1) operation.
- Partition indices by the first byte of records starting with a cryptographic hash.
- Compress the data of each partition with ZStandard [17] level 3 compression.

Figure 9 presents C-style definitions of data structures in the most commonly used indices:

- block_meta index contains the metadata about each blockchain block.
- txo index contains asset amounts for each transaction output and the stake account it affects.

- txo_use index contains the offsets of spending transactions and an epoch to simplify partitioning during the preparation of per-epoch per-account balance changes described in Section 4.3.
- account_update structure is used in Inflows and Outflows temporary indices used to prepare the per-epoch per-account balance changes.
- vrf index is used to collect the information necessary for the delayed processing of VRF nonce updates and the verification of the block leadership eligibility.
- timed_update is a general data structure used to pass updates that either require order or do not need pre-aggregation.

C BENCHMARKING ENVIRONMENT

C.1 Hardware

A dedicated server procurable at Vultr [12] was used in the following configuration:

- AMD EPYC 7443P processor with 24 cores and 48 threads;
- 256GB of RAM;
- 2 NVMe SSDs of 1.92TB each combined into a software RAID0 using Linux's mdadm.

C.2 Operating System

Ubuntu Linux 22.04 LTS was used with the following additionally installed packages:

- docker-compose
- node.js 20

The maximum number of allowed open files was set to 32768 using bash ulimit command.

The source code of the benchmarks is available in the project's GitHub repository:

- End-to-end synchronization [4]
- Incremental synchronization [5]

C.3 Data

Cardano's mainnet data was synchronized up to the slot number 115948800, corresponding to the end of the mainnet's epoch 465.

D CARDANO NODE CONFIGURATION

Cardano Node version 8.1.2 as provided by the official Docker image inputoutput/cardano-node:8.1.2 was used. The source code of the benchmark script is available in the project's GitHub repository [2].

E MITHRIL CONFIGURATION

Mithril was compiled from the source code in its official GitHub repository [9] using the following commands:

```
apt install -y libssl-dev m4
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
git clone https://github.com/input-output-hk/mithril.git
git checkout d711970df9d6be88a14c4f5898d5527aa7c11ded
cd mihtirl-client-cli
make -j build
```

The source code of the benchmark script is available in the project's GitHub repository [3].

revised 19 February 2024