SChain: A Scalable Consortium Blockchain Exploiting Intra- and Inter-Block Concurrency

Zhihao Chen[1] Haizhen Zhuo[2] Quanqing Xu[2] Xiaodong Qi[1] Chengyu Zhu[1] Zhao Zhang[1]
Cheqing Jin[1,*] Aoying Zhou[1] Ying Yan[2] Hui Zhang[2]
[1]East China Normal University [2]Ant Group
{chenzh, xdqi, cyzhu}@stu.ecnu.edu.cn, {zhzhang, cqjin, ayzhou}@dase.ecnu.edu.cn
{haizhen.zhz, xuquanqing.xqq, fuying.yy, shengchu.zh}@antgroup.com

ABSTRACT

We demonstrate SChain, a consortium blockchain that scales transaction processing to support large-scale enterprise applications. The unique advantage of SChain stems from the exploitation of both intra- and inter-block concurrency. The intra-block concurrency not only takes advantage of the multi-core processor on a single peer but also leverages the capacity of multiple peers. The inter-block concurrency enables simultaneous processing across multiple blocks to increase the utilization of various peers. In our demonstration, we use real-time dashboards containing visualization based on the output of SChain to give the attendees interactive explorations of how SChain achieves intra- and inter-block concurrency.

PVLDB Reference Format:
Zhihao Chen, Haizhen Zhuo, Quanqing Xu, Xiaodong Qi, Chengyu Zhu, Zhao Zhang, Cheqing Jin, Aoying Zhou, Ying Yan, Hui Zhang. SChain: A Scalable Consortium Blockchain Exploiting Intra- and Inter-Block Concurrency. PVLDB, 14(12): 2799 - 2802, 2021.
doi:10.14778/3476311.3476348

1 INTRODUCTION

Consortium blockchain is being widely applied to support large-scale businesses in enterprise collaboration, e.g., the AntChain [1] has been conducted with more than 50 multilateral collaboration scenarios. We exemplify one typical scenario about supply chain finance in the upper part of Figure 1. In such a scenario, multiple participants such as banks, insurance & trust companies cooperate to host the blockchain network, each of which commonly devotes multiple peers. Observe that peers belonging to different participants are mutually distrusting due to the potentially hostile environment, while ones within the same participant have a trust foundation. Motivated by this fact, we want to scale the system in terms of each participant to support extensive applications.

To empower the individual participant, a general option is to replace the sequential execution mechanism with a concurrent one [2, 7, 8] to leverage the modern multi-core processors. Fabric [3] introduces the concurrency to blockchain under an *execute-order-validate-commit* paradigm, where transactions are executed in parallel before ordering. In this paradigm, however, the execution

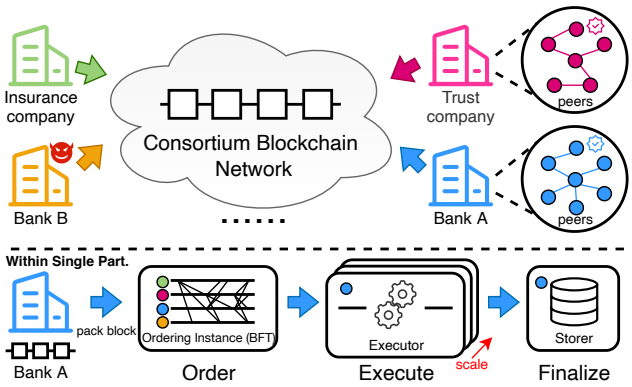

Figure 1: Example scenario of the consortium blockchain

results are then validated sequentially after ordering to abort transactions that cause violation of serializability. Although FastFabric [5] parallelizes the validation for transactions across blocks, it still inherits the limitation that read/write sets are validated sequentially. On the other hand, ParBlockchain [2] and BlockchainDB [8] first analyze the conflicts among transactions and then allow non-conflicting transactions to be executed in parallel. However, current solutions have fundamental limitations from two aspects. (1) These works only arrange a single peer to execute all transactions. When a peer is dedicated fully, the performance cannot be further scaled. (2) The concurrent execution merely involves transactions within a block where transactions of later blocks cannot be executed before the execution of the current block terminates. This approach does not consider the transaction parallelism across multiple blocks.

To address these issues, we present a scalable blockchain system *SChain*, which leverages both *intra-block* and *inter-block* concurrency to scale the transaction processing. In general, SChain follows a novel *scalable order-execute-finalize* (𝕊OEF) paradigm as illustrated from the view of a single participant at the bottom of Figure 1. The transaction flow is divided into three phases: *ordering*, *execution* and *finalization*, which can run on different peers. For intra-block concurrency, transactions within a block are distributed over multiple executors for parallel execution to break through the performance bottleneck of a single peer. Moreover, each executor employs deterministic concurrency control to enable concurrent transaction execution locally. For inter-block concurrency, SChain pipelines the processes of blocks and therefore allows transactions across multiple blocks to execute concurrently. This approach overcomes the existing block-by-block execution drawbacks and enables executors to execute transactions constantly, further improving the resource utilization of different peers.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476348

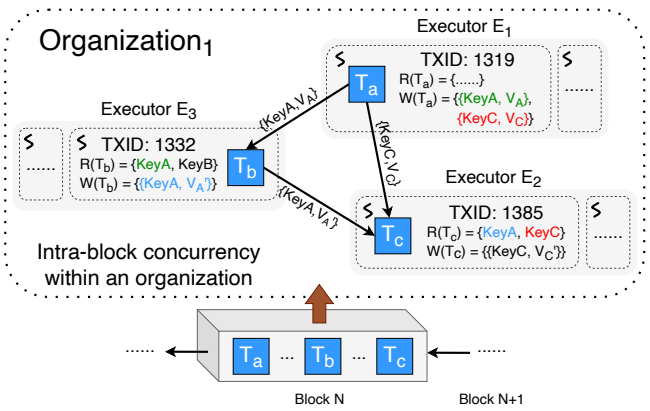

Figure 2: Exploiting intra-block concurrency through distributed concurrent execution mechanism

The purpose of this demonstration is to show the attendees impressive improvement and common usability of SChain. More specifically, we manage to provide an interactive front-end UI for attendees to operate with SChain. The attendees will gain both a high-level overview and a deep insight into the unique techniques utilized by SChain. In our demonstration, Section 2 gives an overview of SChain, introducing its core techniques and architecture. In Section 3, the demonstration details are presented with an interactive user experience of our system.

2 SCHAIN OVERVIEW

2.1 Threat Model

SChain operates under hybrid trust and fault assumptions. Peers are grouped into organizations. An organization here is regarded as an individual participant. Recall that peers within the same participant trust each other while others share no trust. Since there is no trust among participants, SChain employs the PBFT protocol to achieve consensus, where at most f participant(s) can be malicious out of n participants such that $n \geq 3f + 1$.

2.2 Intra- and Inter-Block Concurrency

Motivated by the assumptions above, SChain extends the concurrency to improve transaction processing. We summarize the key features of SChain as the intra- and inter-block concurrency with further discussions about them.

2.2.1 ***Intra-Block Concurrency.*** SChain proposes a distributed concurrent execution mechanism to enhance the capacity of a single participant. That is, the intra-block transactions are distributed to multiple executors for concurrent execution. In intuition, transactions are executed in parallel among executors. Moreover, this mechanism ensures they are executed concurrently within a single executor and the merge of their results is deterministic. Then we illustrate the design details.

To maximize the parallelism among executors, SChain analyzes all potential conflicts among transactions and dispatches utmost conflicting transactions to the same executor. Then, each executor executes received transactions concurrently following the deterministic concurrency control (DCC) [4, 9]. The DCC protocol enables transactions to execute concurrently while still promising the execution result is equivalent to the serial order determined by ordering. Additionally, the protocol requires the read/write keys of transactions should be known in advance by some techniques. SChain holds this acquisition for Turing-complete smart contracts by using static analysis and speculative execution which is also adopted by ParBlockchain [2].

Ideally, transactions share no conflicts among executors so that executors can execute them locally and concurrently following the DCC protocol. However, such a division is hard to guarantee under high contention. Further considering workload balance, SChain allows conflicting transactions to be executed by different executors. While SChain still guarantees that the merge of the execution results is equivalent to the predetermined serial order. This is achieved by migrating the outputs of transactions among executors. In particular, an executor E will not execute a transaction that reads the output of another transaction assigned to executor E' before E' correctly forwards the output to E.

Since peers within the same participant trust each other, the execution among them assumes no Byzantine behavior. We have omitted details relating to aborts and resilience to crash faults. These details are discussed in the extended version.

Example 2.2.1. Figure 2 details how SChain applies the proposed mechanism to realize high concurrency and workload balance in contention. Suppose there are three executors E_1, E_2 and E_3 belonging to $Organization_1$. SChain is currently processing the block N which has already been ordered. Then transactions of block N are partitioned and sent to executors. For instance, T_a and T_b are assigned to different executors while they share a write-read dependency on $KeyA$. SChain guarantees that E_3 can only execute T_b until E_1 forwards the output of $KeyA$ to E_3. This is achieved by analyzing their predetermined $TXID(s)$ and the keys of read/write sets before division. On the other hand, T_c should wait for both the output of T_a and T_b. The migration of data ensures the merged result of all executors is serializable and deterministic. We emphasize that this behavior will not interfere with the concurrent execution of other non-conflicting transactions.

2.2.2 ***Inter-Block Concurrency.*** Different from existing works, the innovation of SChain is to explore the potential of inter-block concurrency. By interleaving the workflows for different blocks, SChain forms the pipelined workflow. Notably, SChain granularly divides its three phases into five stages as presented in Figure 3. Then these stages can be conducted on different peers so that the stages across blocks may overlap. Benefit from this design, SChain can simultaneously process multiple blocks. Thus, its workflow is no longer block-by-block quiescently and all types of peers can keep on working, fully utilizing the system resources. Furthermore, SChain allows transactions in later blocks to be executed earlier if they do not conflict with executing transactions. This design ensures inter-block transactions execute concurrently among all executors, leading to the *transaction streaming pipeline*. It means transactions across multiple blocks are flowing among stages constantly to finish their execution. It also promises the worker threads on executors always have enough transactions to execute, further improving the overall performance.

Example 2.2.2. Figure 3 depicts how the transaction streaming pipeline empowers SChain to achieve inter-block concurrency over

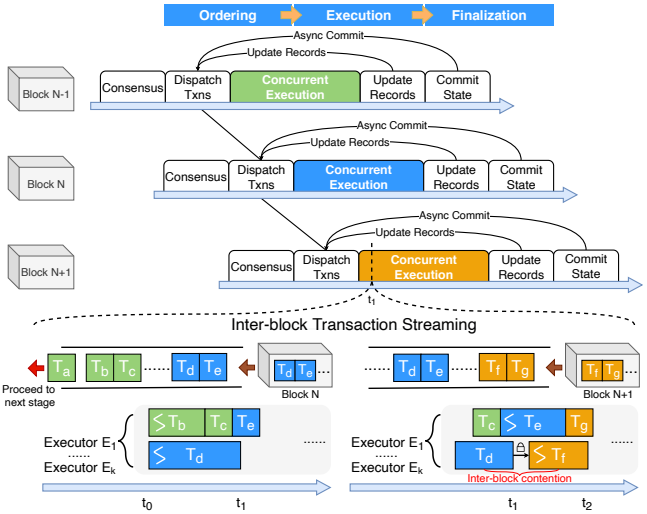

Figure 3: Exploiting inter-block concurrency through transaction streaming pipeline

three blocks. In the upper part of the figure, three pipes of $Block_{N-1}$, $Block_N$ and $Block_{N+1}$ form the pipelined workflow. As can be seen, the workflows for different blocks interleave. By obeying the mentioned rules, a transaction in $Block_{N+1}$ can execute before some transactions in $Block_N$. To dive into this process, we pick up a timepoint t_1 and reveal corresponding details in the bottom part. In the left bottom, worker threads on executor E_1 are concurrently executing T_b of $Block_{N-1}$ and T_d of $Block_N$ at timepoint t_1. Once the execution stage of $Block_{N+1}$ starts, belonging transactions will enter the streaming processing. In the right bottom, T_g of $Block_{N+1}$ shares no conflicts so that it can be executed in parallel with others once T_e finishes its execution to set an idle worker thread. On the other hand, T_d will apply the output of records to activate the execution of T_f since T_f conflicts with it. Benefit from the transaction streaming pipeline, transactions across multiple blocks can simultaneously execute to exploit the inter-block concurrency.

2.3 Architecture

We design the SChain's architecture based on proposed techniques. As shown in Figure 4, SChain consists of multiple organizations that form the network. The transactions sent by clients will go through three phases of processing: ordering, execution and finalization.

Then we illustrate how SChain operates in these three phases. First, the ordering phase establishes a global order on all transactions provided by clients through consensus. After ordering, these transactions are distributed over executors within each organization for execution. SChain applies the distributed concurrent execution and transaction streaming pipeline during the execution phase. Last, the finalization phase collects the updates of records and commits them.

The decomposed phases of SChain provide great flexibility. For instance, the ordering phase can scale easily by running concurrent instances [6] to compromise the global order. Considering this demonstration does not focus on scaling the ordering, we thus omit the implementation details. Besides, SChain avoids distributed

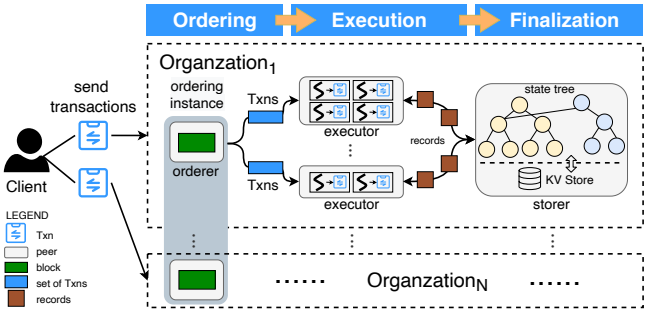

Figure 4: SChain Architecture

transactions because each transaction is handled by a unique executor and all relevant data is migrated to this executor. Thus, each organization can devote more executors on-demand to raise the parallelism and processing of execution.

3 DEMONSTRATION

We deploy SChain in the back-end and demonstrate it in two scenarios. An interactive front-end interface is provided to help the attendees step through the process of intra-block execution and inter-block workflow. Additionally, we showcase the scalability of execution through a graphical dashboard that displays the detailed system performance.

3.1 Setup and Interface

The demo SChain is initially deployed up to four organizations, each of which may jointly run many ordering instances and deploy multiple executors. We monitor and interact with one target organization and present its information in Figure 5, which are divided into five panels A~E. Panel A and B present the latest information on blocks and transactions. Panel C~E depict the runtime status of the orderers, executors and storer respectively.

Batch Invoke Interface. At the beginning of the demonstration, the attendees will send transactions to start the process. We provide a user-friendly interface called *batch invoke* to eliminate the complexity of invoking smart contracts through pre-installing a CPU-heavy smart contract on SChain. By using this interface, the front-end will automatically sign and send transactions referring to this smart contract. Note that the pre-acquisition keys needed by SChain are also included in the transaction. Optional inputs are provided for attendees to determine the batch size and additional transaction payloads, as shown on Panel F of Figure 6.

3.2 Scenario 1: Gain Insight into Intra-Block Concurrency

In the first scenario, the attendees have a chance to dive into the distributed concurrent execution. They use the batch invoke interface to send transactions by filling the batch size and optionally extra data. Once transactions are ordered and distributed over executors, the system dashboard in Figure 5 will visualize the process. The attendees can click the "View all transactions" on panel B to view the status of various concurrently executing transactions. The runtime results for transactions are shown on panel G of Figure 6.

Records are transferred among executors to guarantee the correctness of intra-block concurrency. To trace the entire flow, the

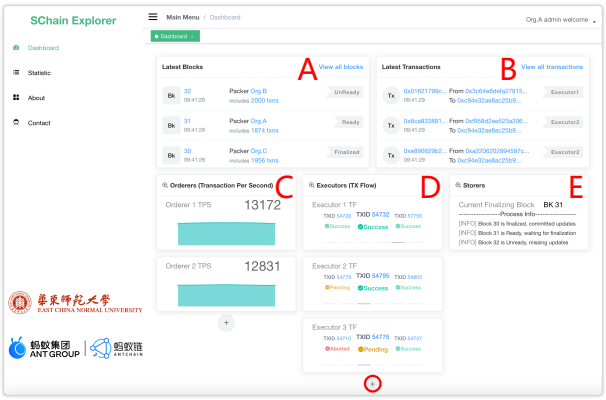

Figure 5: System dashboard

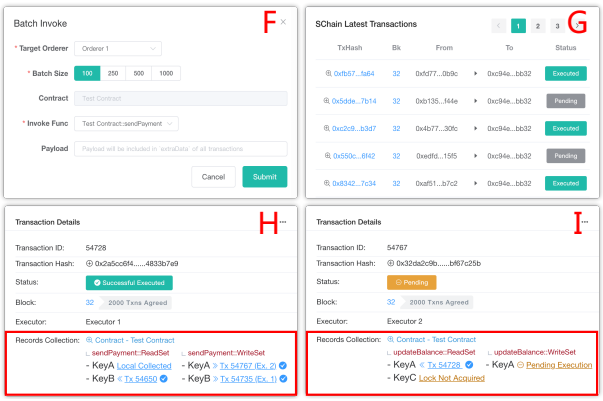

Figure 6: Insight of intra-block concurrency

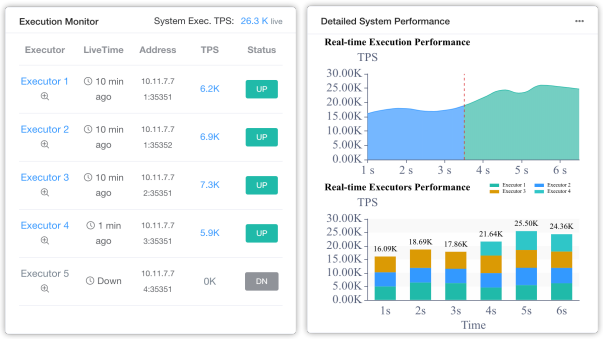

Figure 7: Showcase the performance of multiple executors

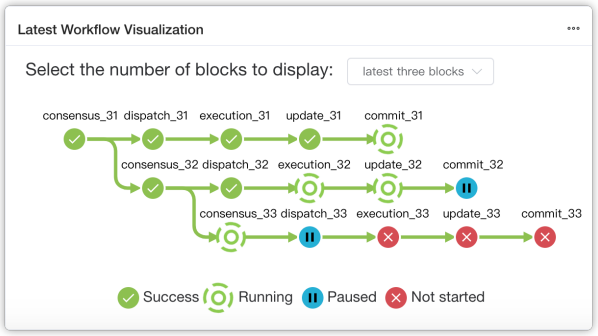

Figure 8: Inter-block concurrency visualization

attendees can further inspect interested transactions. Panel H and I reveal two transactions in detail. The "Records Collection" on each panel shows the status of records. For instance, the transaction on panel H has collected all the records in its read-set: *KeyA* and *KeyB*. After being executed by E_1 under the DCC protocol, the output of record *KeyA* will be transferred to executor E_2 since it is required by the transaction on panel I. This migration of data is recorded to show the attendees how data is driven from the insight.

Furthermore, the attendees will appreciate the scalability of execution. If the ordering performance fully covers the execution, we allow attendees to scale the execution on-demand. They can click the red button in the middle bottom of Figure 5 to add an executor. Note that the newly added executor takes effect in the next block height. As a result, the attendees will wait a moment to see the improvement in Figure 7 after adding an executor.

3.3 Scenario 2: Investigate Inter-Block Concurrency

This scenario gives the attendees explorations of inter-block concurrency in SChain and sheds light on its distinguished pipeline design. The graphical interface allows the attendees to select the number of pipes to display. They will view the simultaneous process across multiple blocks after selection. For example, in Figure 8, the latest three pipes are selected to display, involving $block_{31}$, $block_{32}$ and $block_{33}$. Stages denoted by circles horizontally form each block's workflow and their runtime status is illustrated at the bottom of Figure 8. Given this investigation, the attendees will better understand how SChain achieves inter-block concurrency by pipelining the workflows across multiple successive blocks.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation of China (U1911203, U1811264 and 61972152).

REFERENCES

[1] 2021. AntChain. https://antchain.antgroup.com/.
[2] M. J. Amiri, D. Agrawal, and A. E. Abbadi. 2019. ParBlockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems. In *ICDCS*. IEEE, 1337–1347.
[3] E. Androulaki, A. Barger, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys*. ACM, 30:1–30:15.
[4] J. M. Faleiro, D. Abadi, and J. M. Hellerstein. 2017. High Performance Transactions via Early Write Visibility. *Proc. VLDB Endow.* 10, 5 (2017), 613–624.
[5] C. Gorenflo, S. Lee, L. Golab, and S. Keshav. 2019. FastFabric: Scaling Hyperledger Fabric to 20, 000 Transactions per Second. In *IEEE ICBC*. IEEE, 455–463.
[6] S. Gupta, J. Hellings, and M. Sadoghi. 2021. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing. In *ICDE*. IEEE, 1392–1403.
[7] C. Jin, S. Pang, X. Qi, Z. Zhang, and A. Zhou. 2021. A High Performance Concurrency Protocol for Smart Contracts of Permissioned Blockchain. *TKDE* (2021), 1–1. https://doi.org/10.1109/TKDE.2021.3059959
[8] S. Nathan, C. G., A. Saraf, M. Sethi, and P. Jayachandran. 2019. Blockchain Meets Database: Design and Implementation of a Blockchain Relational Database. *Proc. VLDB Endow.* 12, 11 (2019), 1539–1552.
[9] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD Conference*. ACM, 1–12.