Deterministic Concurrency Control For Private Blockchains

Ziliang Lai (Supervisor: Eric Lo)
The Chinese University of Hong Kong
{zllai,ericlo}@cse.cuhk.edu.hk

1 PROBLEM AND MOTIVATION

Private blockchain has gained popularity in inter-organization business processing (e.g., supply chain management [15] and cross-border payment [9]). It allows mutually untrusted parties to share data and mutate data in an endorsed manner. A typical private blockchain is composed of a *consensus layer* pipelined with an *execution layer*, where the organizations agree on the blocks of transactions using the consensus layer and the transaction blocks are then passed to each organization's executor for execution. To ensure the organizations share an identical view of data, one core requirement of the executor is *determinism*, i.e., given the same transaction blocks and the same initial state, the executors must produce an identical new state.

While the consensus layer was regarded to be the major bottleneck of the blockchains, the relaxed requirement in the inter-organization setting (e.g., static membership, stable network connection) enables recent optimizations [3, 10, 23] to achieve astonishingly good performance, which in turn shifts the bottleneck to the execution layer [1, 20]. However, even the latest research in private blockchain offers unsatisfactory throughput in the execution layer due to the requirement of determinism. That's because determinism limits the parallelism of execution since parallel execution often results in non-deterministic execution order. Therefore, recent works cannot execute transactions with full concurrency [1, 14, 18–20]. In this paper, we present Harmony, a deterministic concurrency control algorithm that is highly concurrent and yields 2.0 × to 3.5 × better throughput over the state-of-the-art competitors. Harmony is built on the shoulder of the latest deterministic databases [7, 8, 12, 21] that optimize concurrency control while upholding determinism. Based on that, Harmony incorporates *inter-block parallelism* and *judicious abort* which is crucial for performance in blockchain setting.

Private Blockchains	Execution Layer	Problem
Quarum [2]	Serial execution	No concurrency
Tendermint [4]		
Fabric [1]	(1) Parallel simulation (2) Serial validation	Limited concurrency
Fabric++ [20]		
Fabric# [18]		
RBC [14]		

Table 1: The problem of the execution layer in existing private blockchains

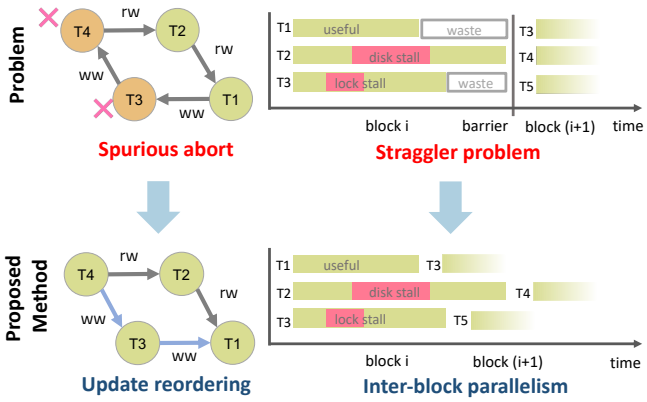


Figure 1: The problems of applying the latest deterministic database to blockchain, and our proposed methods

2 BACKGROUND AND RELATED WORK

2.1 Private Blockchains

Besides the work on optimizing the consensus layer [3, 6, 10, 11, 23, 25], the importance of the concurrency control in the execution layer has also been noticed [1, 14, 18–20]. Table 1 shows a summary of some representatives.

Serial execution is a straightforward method for ensuring determinism. Given the same transaction block generated from the consensus layer, the executor in Quorum [2] and Tendermint [4] executes the transactions serially following the order of their transaction IDs (`TID`s). Assuming the transactions contain no non-deterministic operations (e.g., `rand()` and `time()`), each executor is guaranteed to yield the same state. However, this method yields limited throughput because it fails to utilize the core-parallelism in modern CPUs.

Some recent private blockchains employ optimistic methods to enable partial parallelism [1, 14, 18, 20]. Specifically, the transactions are executed in a Simulation phase and a Validation phase. The Simulation phase is able to execute transactions optimistically in parallel, while the Validation phase has to validate serializability (i.e., the concurrent execution is equivalent to a serial execution) *in a serial manner* to uphold determinism. Therefore, the degree of concurrency is limited.

2.2 Deterministic Databases

While private blockchains have started to pursue concurrent transaction execution recently, deterministic databases [7, 8, 12, 21] have already studied deterministic concurrency control for almost a decade. Astonishingly, the connection between private blockchain and deterministic database has never been explicit, although the connection between the former and distributed database has been

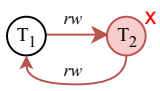
studied elsewhere [17]. Private blockchain and deterministic database actually share a lot in common. First, both of them are distributed in nature. Second, they both process transactions block-by-block. Third, both blockchains and deterministic databases require determinism.

One reason that impedes deterministic databases to be adopted to private blockchains is the strong assumptions required by the classic deterministic databases. They assume every transaction's read-write sets can be inferred before execution [7, 8, 21], but that is not generally applicable (e.g., when the transaction contains branches based on the query results). Nonetheless, the latest deterministic database Aria [12] eliminates this requirement by executing transactions in a Simulation phase and a Validation phase similar to the recent private blockchains. However, Aria is more advanced than private blockchains by offering *concurrency in both phases*.

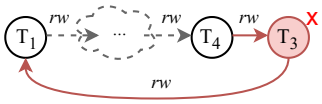
Given a block of transactions, Aria generates *deterministic* read-write sets of the transactions in the Simulation phase. Taking the read-write sets as input, its Validation phase validates serializability by examine *dangerous dependency patterns* and commits transactions deterministically in parallel. Concretely, in the Simulation phase, Aria executes the transactions against a *block snapshot*, which is the state after processing the previous block. In this phase, no modification is applied, and each partition only buffers its write-set in its private workspace. With the same block snapshot and the same transaction blocks, executors in Aria generates identical read-write sets. In the Validation phase, the dependencies among transactions can be identified. For example, if two transactions T_1 and T_2 have overlapping write-sets, there is a write(w)-write(w)-dependency among then. Aria deterministically abort some transactions to uphold serializability by examine the dangerous dependency patterns. For example, the ww-dependency is one of the dangerous dependency patterns in Aria. On seeing T_1 ww-depending on T_2, Aria aborts the one with the larger `TID` (i.e., T_2). By validating based on the deterministic read-write sets and the `TID`s, a deterministic set of the transactions is aborted in this phase. Moreover, since the transactions that passes the validation have non-overlapping write-sets, they can apply their updates in parallel without breaking determinism.

However, even the latest deterministic database is not a perfect fit for private blockchain because of the difference in design choices. Specifically, Aria is designed for main-memory databases, while private blockchains are commonly disk oriented. That's because disk-based storage is more cost effective and upgrading to main-memory design does not payoff since the consensus layer in the pipeline cannot saturate the in-memory executor. Specifically, directly porting Aria's deterministic concurrency control to disk oriented private blockchains causes two major problems as summarized in Figure 1.

- **Spurious Aborts.** Aborting a transaction on seeing a ww-dependency is overly conservative because two transactions T_1 and T_2 that update the same item x can still form a serializable schedule. While spurious aborts are acceptable in Aria because retrying an aborted transaction in main-memory databases is cheap – which is not the case in private blockchains because retrying a transaction means going through consensus and disk I/Os again.



(a) A match with only two transactions



(b) A match with at least three transactions

Figure 2: Examples of backward dangerous structure

- **Stragglers.** Aria has strict barriers between blocks, i.e., only after block i has finished execution, block $(i+1)$ can start, which causes the straggler problem. Due to disk stalls, the variance of the transaction runtime in private blockchains is higher than main-memory databases, and thus the straggler problem is exaggerated.

3 UNIQUENESS AND APPROACH

3.1 Fusing Deterministic Database With Private Blokchain

Insight. Given the close connections, private blockchains can actually absorb many techniques from deterministic databases. In particular, the advanced deterministic concurrency control can be adopted to improve the concurrency in private blockchains.

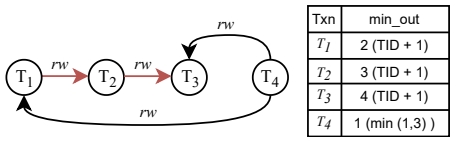
Approach. We port the deterministic concurrency control algorithm in Aria to BCR [14], a private blockchain that builts on top of PostgreSQL [13], which allows blockchains to support full-fledged SQL functionalities like a relational database. To mitigate Aria's downsides when applied to blockchain (i.e., spurious aborts and straggles) we propose a new deterministic concurrency control Harmony, which incorporates blockchain-specific optimizations (see below).

3.2 Judicious Validation

Insight. The idea of dangerous dependency pattern enables transactions to validate independently in parallel because they only examine dependencies that is "local" to themselves. However, a pattern based on a single dependency (e.g., ww-dependency in Aria) could cause many spurious aborts. Harmony design a new dangerous patterns based on a *pair* "local" dependencies, such that fewer transactions match the pattern and abort, while the level of concurrency is almost unaffected.

Approach. Harmony abort a transaction if it matches a *backward dangerous structure*. Specifically, T_j is aborted if there exists $T_i \xleftarrow{rw} T_j \xleftarrow{rw} T_k$, $i < j$ and $i \leq k$, where we use $\xleftarrow{rw}$ to represent read(r)-write(w)-dependencies.

Figure 2 shows two examples that match the backward dangerous structure. Notice that a backward dangerous structure can be as small as having two transactions only since we allow $i = k$ (Figure 2a). A dangerous structure can also involve an arbitrary number of transactions (e.g., Figure 2b, where $T_1 \xleftarrow{rw} T_3 \xleftarrow{rw} T_4$). Overall, by eliminating all backward dangerous structures, Harmony eliminates cycles in the rw-dependency graph. Nonetheless, all the cycles in the compelte dependency graph must be eliminated to achieve serializability. Harmony offloads the ww-dependencies to *update reordering* such that the whole dependency graph is acyclic.



Txn	min_out
T_1	2 (TID + 1)
T_2	3 (TID + 1)
T_3	4 (TID + 1)
T_4	1 (min (1,3))

Figure 3: An example of update reordering

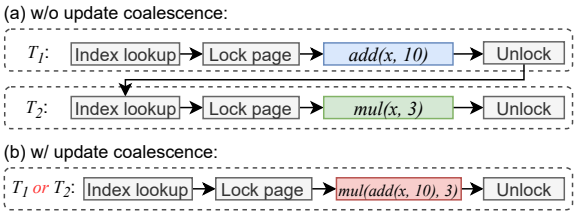


Figure 4: Physical plans of the update commands with and without coalescence

3.3 Update Reordering

Insight. Given an acyclic rw-dependency graph, reordering the ww-dependencies based on the topological order of the rw-dependency graph allows the complete dependency graph to be acyclic. As the example shown in Figure 1, by reordering the ww-dependencies following the direction of the rw-dependencies, the backward edges are eliminated and thus cycles are avoided in the dependency graph.

Approach. Although useful, the topological sort is expensive and hard to parallelize. We exploit the property of the acyclic rw-dependency graph (i.e., all backward dangerous structures are removed) to convert the topological sort into *quick-sorts*.

Figure 3 shows an example graph without backward dangerous structures. A topological sort on it results in $[T_4, T_1, T_2, T_3]$. Harmony computes a `min_out` for each transaction T representing the minimum `TID` in T's outgoing edges. Suppose only T_2 and T_4 update x, Harmony only needs to quick-sort the `min_outs` of T_2 and T_4 without traversing the whole graph like topological sort. Nonetheless, the resulting order (i.e., $[T_4, T_2]$) is consistent with the topological ordering. Besides, if T_1, T_2 and T_3 update y, the two updater lists (i.e., $[T_2, T_4]$ on x and $[T_1, T_2, T_3]$ on y) can be sorted in parallel.

3.4 Update Coalescence

Insight. The physical plans of the update operations on the same record largely overlap and they can actually be merged into one operation. Figure 4 shows an example, where the physical plans of two updates on x actually share the same index lockup and locking operations.

Approach. As shown in Figure 4, Harmony merges the overlapped parts of the physical plans on updating the same record, while the order of updates posed by the update reordering is respected. The coalesced update operation only needs to be performed by one of the transactions (i.e., T_1 or T_2), and the other transaction can skip and perform other updates in parallel.

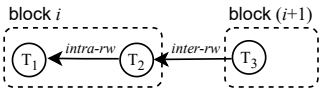


Figure 5: An example that may cause non-determinism if without the Enhanced Validation Rule

3.5 Inter-block Parallelism

Insight. The reason behind the straggler problem is the barrier between blocks. As shown in Figure 1, Harmony breaks the block barrier and allows adjacent blocks to run concurrently, which can improve the resource utilization.

Approach. Inter-block parallelism may induce inter-block dependencies, and the the judicious validation has to be enhanced to handle them. Moreover, inter-block dependencies may cause non-determinism due to network asynchrony. For example, Figure 5 shows a backward dangerous structure across blocks. The executor in the organization O1 who sees the whole backward dangerous structure would abort T_2. However, if block $(i+1)$ is delayed due to network asynchrony in organization O2, its executor would commit T_2 because when examining the backward dangerous structures, the executor of O2 only sees $T_1 \xleftarrow{intra-rw} T_2$. This causes inconsistency between $O1$ and $O2$. Harmony breaks the backward dangerous structure across blocks by always aborting the transactions in the later blocks (i.e., T_3 in the example). This ensures all the executors abort the same transaction regardless of the network delay.

4 RESULTS AND CONTRIBUTIONS

4.1 Performance Comparisons

We perform experiments on machines with 2.1GHz Intel Xeon E5-2620v4 with 64GB DRAM and 800GB SSD, running 64-bit CentOS 7.6 with Linux Kernel 3.10.0 and GCC 4.8.5. Nodes are connected using 1Gbps Ethernet. Harmony is compared with state-of-the-art blockchains: Fabric [1], FastFabric [18], and RBC [14]. Besides, to demonstrate the effectiveness of our blockchain-specific optimizations, we also compared with AriaBC, i.e., the blockchain that we built by directly porting the deterministic concurrency control in Aria to RBC [14]. There are four organizations in our experiment and we follow the recent works [1, 14, 18, 20] to use Kafka as the consensus layer.

We use both YCSB [5] and Smallbank [16] as the benchmarks. For YCSB, we set the number of keys to 10k and follow [12, 22, 24] to wrap 10 operations into one transaction. Each operation has equal probabilities of being a read (`SELECT`) or a write (`UPDATE`). For Smallbank, we also set the number of accounts to 10k and use the standard mix. Both benchmarks have a skewness of 0.6 by default to create medium contention.

We measure the peak throughput and the end-to-end latency of the committed transactions for all systems. Figures 6 and 7 show the results. HarmonyBC attains 3.5× and 2.0× throughput over the best of the existing private blockchains (i.e., RBC in this experiment) in Smallbank and YCSB, respectively. It also achieves the latency comparably good as the best of the state-of-the-art.

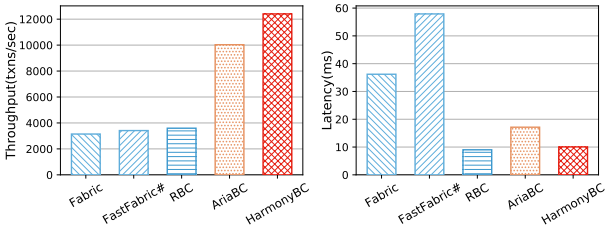


Figure 6: Overall performance on Smallbank

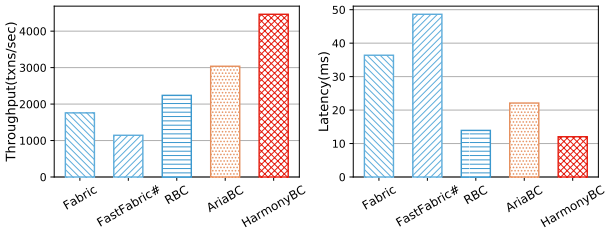


Figure 7: Overall performance on YCSB

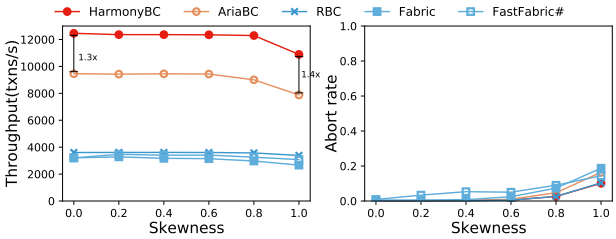


Figure 8: Impact of contention on Smallbank

Simply implementing Aria as a private blockchain (i.e., AriaBC) readily yields better throughput than existing private blockchains, which shows the benefits of using an advanced deterministic concurrency control from a deterministic database to improve private blockchain. Harmony attains a larger margin (e.g., 1.5× throughput over AriaBC in YCSB), demonstrating the effectiveness of HarmonyBC's blockchain-specific optimizations.

To further compare the performance under different contentions, we vary the skewness of the accessed data to control the degree of contention. As shown in Figures 8 and 9, all systems incur more aborts with a larger skewness and thus their performances drop (less severe in Smallbank since it generally has lower contention compared to YCSB). But, Harmony outperforms its competitors under all skewnesses and it consistently has lower abort rates.

4.2 Contributions

This paper makes the following major contributions.

- We take the first step to explicitly connect private blockchain with deterministic database. By exploiting the connections, we are able to fast-forward the development of private blockchain using lessons learned from deterministic databases.
- Based on the state-of-the-art deterministic concurrency control in the latest deterministic database, we proposed effective techniques to further boosts its performance in blockchain setting.
- Harmony is extensively evaluated using benchmarks commonly used in blockchains including YCSB [5] and SmallBank [16]. Empirical results show that Harmony achieves 2.0× to 3.5× throughput better than RBC [14] and FastFabric# [18], and 2.3× throughput better than AriaBC (the blockchain implemented using the same framework as Harmony but using Aria) under high contention.

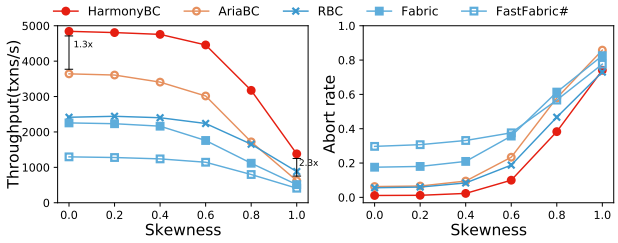


Figure 9: Impact of contention on YCSB

REFERENCES

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*. 1–15.

[2] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. 2018. Performance evaluation of the quorum blockchain platform. *arXiv preprint arXiv:1809.03421* (2018).

[3] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. 2014. State machine replication for the masses with BFT-SMART. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 355–362.

[4] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains*. Ph.D. Dissertation.

[5] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.

[6] James A. Cowling, Daniel S. Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. 2006. HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance. In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, Brian N. Bershad and Jeffrey C. Mogul (Eds.). USENIX Association, 177–190. http://www.usenix.org/events/osdi06/tech/cowling.html

[7] Jose M Faleiro and Daniel J Abadi. 2015. Rethinking serializable multiversion concurrency control. *Proceedings of the VLDB Endowment* 8, 11 (2015).

[8] Jose M Faleiro, Daniel J Abadi, and Joseph M Hellerstein. 2017. High performance transactions via early write visibility. *Proceedings of the VLDB Endowment* 10, 5 (2017).

[9] Ye Guo and Chen Liang. 2016. Blockchain application and outlook in the banking industry. *Financial innovation* 2, 1 (2016), 1–12.

[10] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. Resilientdb: Global scale resilient blockchain fabric. *arXiv preprint arXiv:2002.00160* (2020).

[11] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. 2007. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, Thomas C. Bressoud and M. Frans Kaashoek (Eds.). ACM, 45–58. https://doi.org/10.1145/1294261.1294267

[12] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. 2020. Aria: a fast and practical deterministic OLTP database. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2047–2060.

[13] Bruce Momjian. 2001. *PostgreSQL: introduction and concepts*. Vol. 192. Addison-Wesley New York.

[14] Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. 2019. Blockchain meets database: Design and implementation of a blockchain relational database. *arXiv preprint arXiv:1903.01919* (2019).

[15] Guido Perboli, Stefano Musso, and Mariangela Rosano. 2018. Blockchain in logistics and supply chain: A lean approach for designing real-world use cases. *Ieee Access* 6 (2018), 62018–62028.

[16] Dan RK Ports and Kevin Grittner. 2012. Serializable snapshot isolation in PostgreSQL. *arXiv preprint arXiv:1208.4179* (2012).

[17] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. 2021. Blockchains vs. Distributed Databases: Dichotomy and Fusion. In *Proceedings of the 2021 International Conference on Management of Data*. 1504–1517.

[18] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A transactional perspective on execute-order-validate blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 543–557.

[19] Felix Schuhknecht, Ankur Sharma, Jens Dittrich, and Divya Agrawal. [n.d.]. chainifyDB: How to get rid of your Blockchain and use your DBMS instead. ([n. d.]).

[20] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*. 105–122.

[21] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 1–12.

[22] Tianzheng Wang and Hideaki Kimura. 2016. Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. *Proceedings of the VLDB Endowment* 10, 2 (2016), 49–60.

[23] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 347–356.

[24] Xiangyao Yu, Andrew Pavlo, Daniel Sanchez, and Srinivas Devadas. 2016. Tictoc: Time traveling optimistic concurrency control. In *Proceedings of the 2016 International Conference on Management of Data*. 1629–1642.

[25] Yunhao Zhang, Srinath T. V. Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine Ordered Consensus without Byzantine Oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 633–649. https://www.usenix.org/conference/osdi20/presentation/zhang-yunhao