Aug 24, 2023
University of Waterloo
Blockchain Meets Deterministic Database (SIGMOD'23)

Bruce Lai, Chris Liu, Eric Lo
Chinese University of Hong Kong

A blockchain is a

- Secure
- Replicated database

A replicated database needs **determinism**

- Same input
- Same output

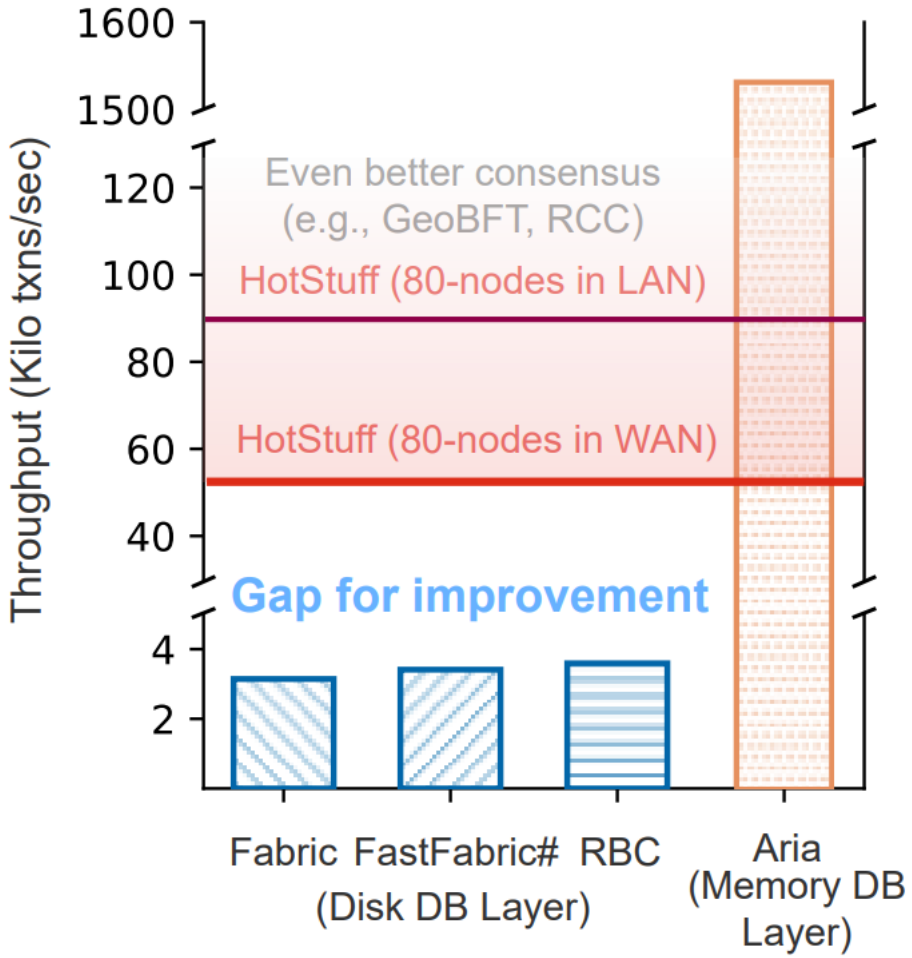
How do blockchains achieve determinism?

- Same input
  - Serial Execution
- Same output

HYPERLEDGER

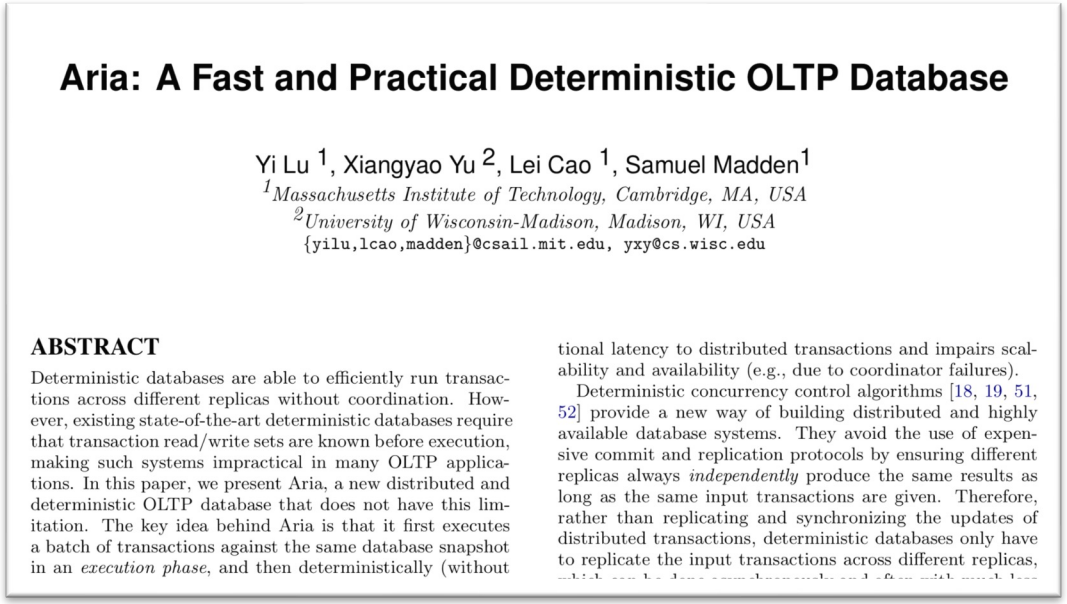
ethereum

Consensus is no longer the bottleneck in private blockchain



Why not use a deterministic database straight?

- "Chainify" PostgreSQL using Aria
- Win hands down



Aria: A Fast and Practical Deterministic OLTP Database

Yi Lu [1], Xiangyao Yu [2], Lei Cao [1], Samuel Madden[1]
[1]*Massachusetts Institute of Technology, Cambridge, MA, USA*
[2]*University of Wisconsin-Madison, Madison, WI, USA*
{yilu,lcao,madden}@csail.mit.edu, yxy@cs.wisc.edu

ABSTRACT

Deterministic databases are able to efficiently run transactions across different replicas without coordination. However, existing state-of-the-art deterministic databases require that transaction read/write sets are known before execution, making such systems impractical in many OLTP applications. In this paper, we present Aria, a new distributed and deterministic OLTP database that does not have this limitation. The key idea behind Aria is that it first executes a batch of transactions against the same database snapshot in an *execution phase*, and then deterministically (without tional latency to distributed transactions and impairs scalability and availability (e.g., due to coordinator failures).

Deterministic concurrency control algorithms [18, 19, 51, 52] provide a new way of building distributed and highly available database systems. They avoid the use of expensive commit and replication protocols by ensuring different replicas always *independently* produce the same results as long as the same input transactions are given. Therefore, rather than replicating and synchronizing the updates of distributed transactions, deterministic databases only have to replicate the input transactions across different replicas,

VLDB2020

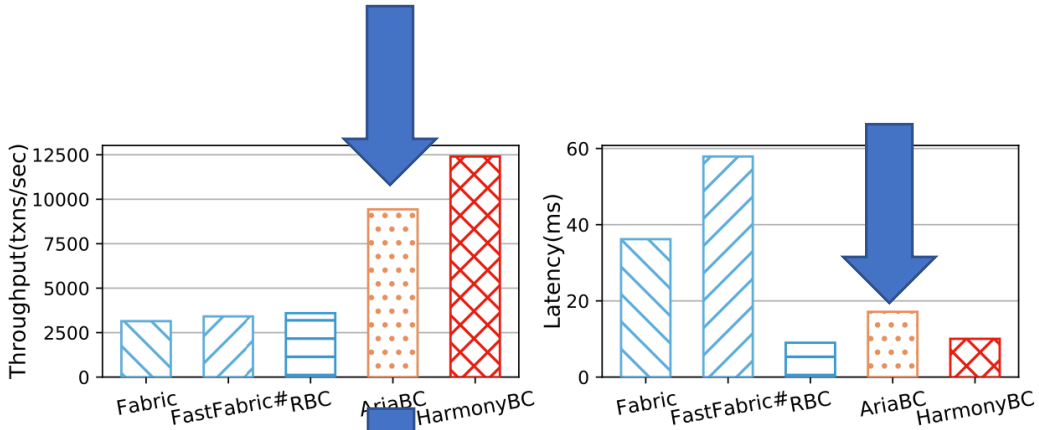


Figure 7: Over l performance on Smallb k

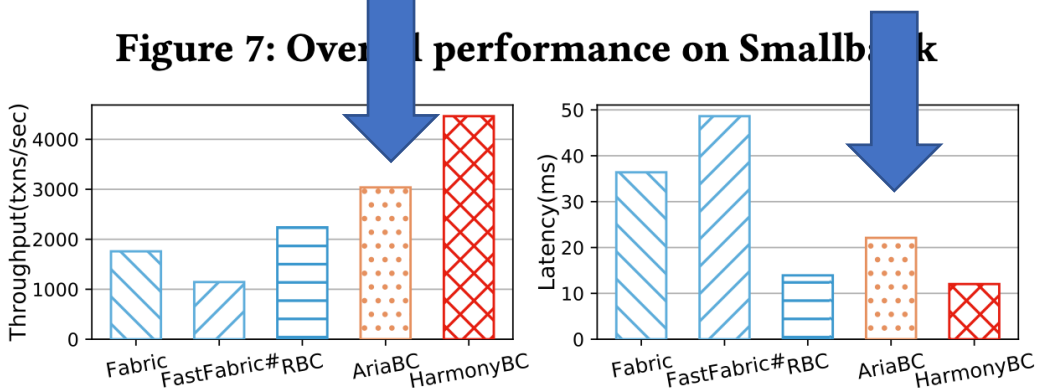
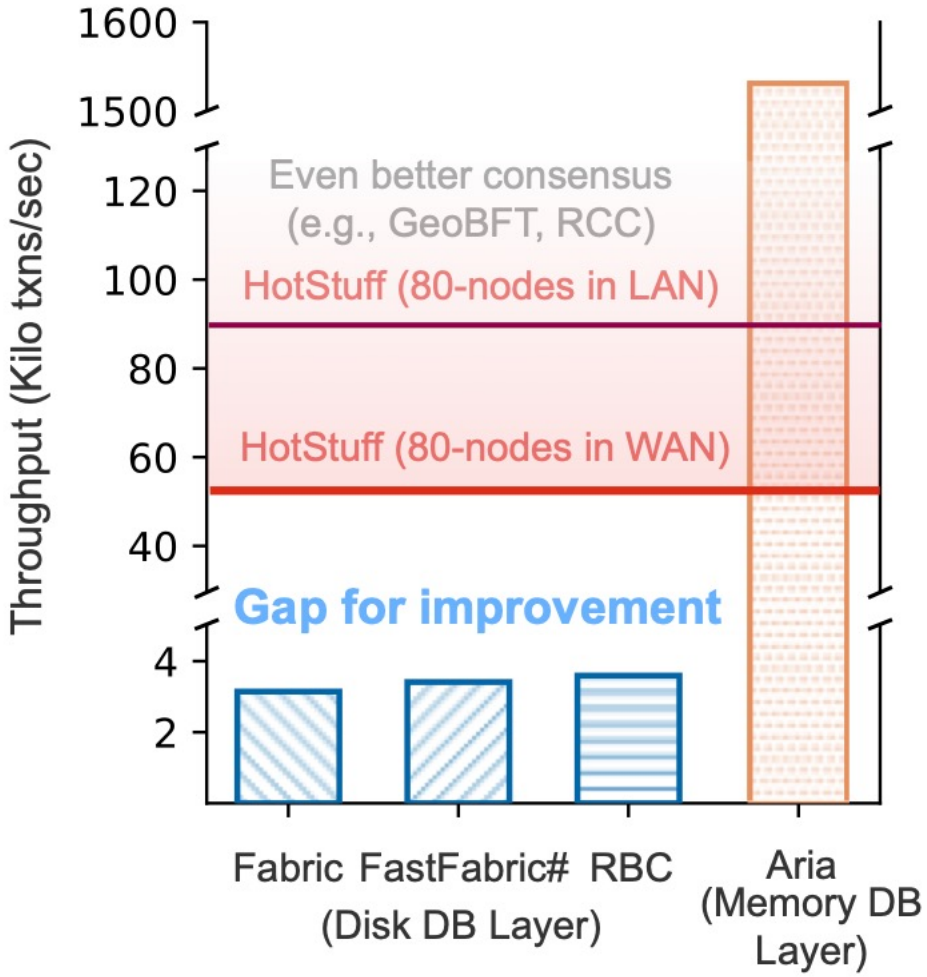


Figure 8: Overall performance on YCSB

Endgame? No

- Deterministic concurrency control (DCC) has been designed for main-memory databases
  - DRAM price (~50USD/GB)



Harmony: Blockchain marries DCC

Harmony

- Deterministic Concurrency Control (DCC) optimized for Disk Blockchain
  - Pessimistic vs Optimistic DCC

Harmony 1: Judicious abort

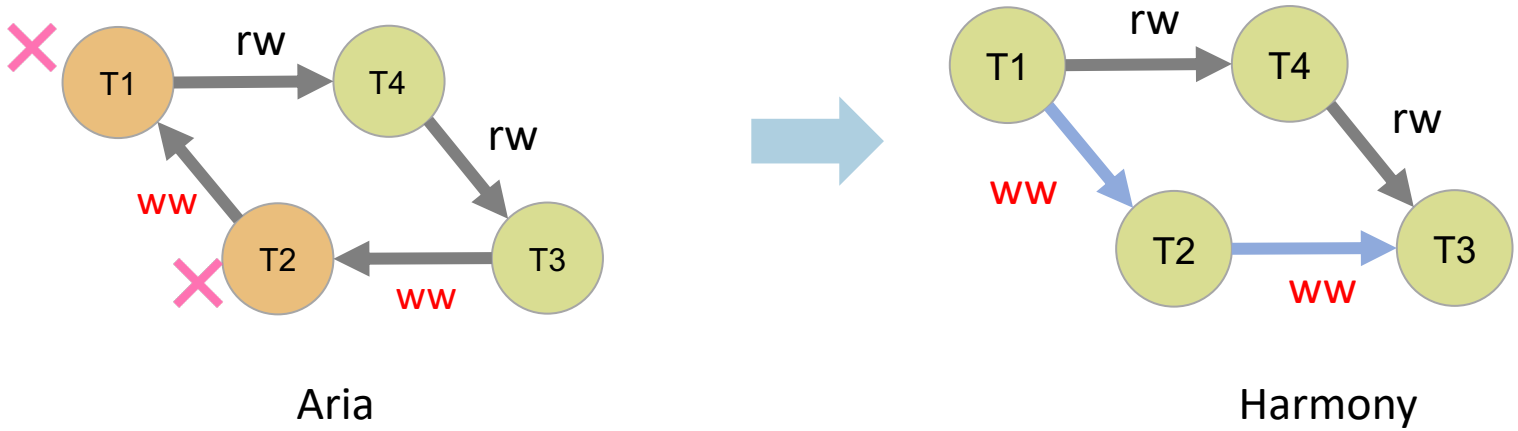
Theory	Aria: aggressive abort	Harmony: judicious abort
1) Cycle detection in dependency graph 2) Break cycle by abort	1) Avoid cycle in **rw dependency subgraph**: Pattern A1: Ti $\xrightarrow{rw}$ Tj $\xrightarrow{rw}$ Tk j > i and j > k **Abort Tj**	1) Avoid cycle in **rw dependency subgraph:** Dangerous backward structure: Ti $\xleftarrow{rw}$ Tj $\xleftarrow{rw}$ Tk i < j and i <= k **Abort Tj** e.g., T2 reads a before-image of T1's write
	2) Avoid cycle in **rw+ww dependency subgraph**: Pattern A2: Ti $\xrightarrow{ww}$ Tj, j > k **Abort Tj**	2) Avoid cycle in **rw+ww dependency subgraph**: ~~Abort~~ -> **Update Reordering**
	3) Complete rw+ww+wr graph: No need to worry wr-dependencies (i.e., dirty reads) because all reads read snapshot from last committed block //i.e., no dirty read by design	
- Expensive unparallelizable cycle detection on the whole graph :(	Lightweight :)	Lightweight :)
No false abort :)	Many false aborts, especially when hotspots (many ww on the hot items) :(	Few false aborts :) Resilient to hotspots :)

Lemma: rw-dependency subgraph is acyclic if transactions in dangerous backward structures are aborted

- Simple Idea: Breaks all **backward (transitive) rw-dependencies**
  - I.e., smaller TID ⟵ higher TID
  - E.g., T1 ⟵(rw) T2 ⟵(rw) T3
- No backward edges? Can't form cycles

No cycle

Next: ensure rw+ww dependency graph is also acyclic



Aria

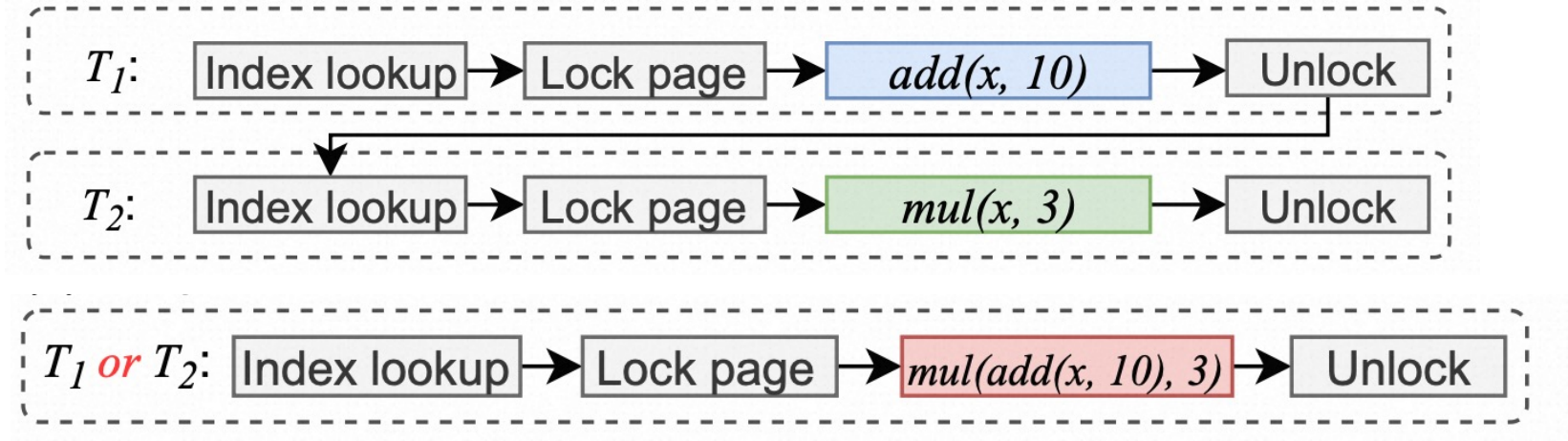
Harmony

On seeing ww-dependency?	
Aria: abort	Harmony: reordering

Harmony 2: Update coalesce (during commit)

Without coalesce:

With coalesce:



Harmony 3: Inter-block Parallelism

	Aria	Harmony
Design Choice	No inter-block parallelism (block i waits block (i-1) to finish)	Inter-block Parallelism (block i can be in parallel with block (i-1))
Rationale	DRAM DB Layer Lower variance in transaction lifespan in a block	I/O DB Layer Higher variance in transaction lifespan in a block => block i-1 has idle cycles
Cost and Benefit	-	Benefit: Better CPU utilization and pipeline Cost: Inter-block dependency tracking

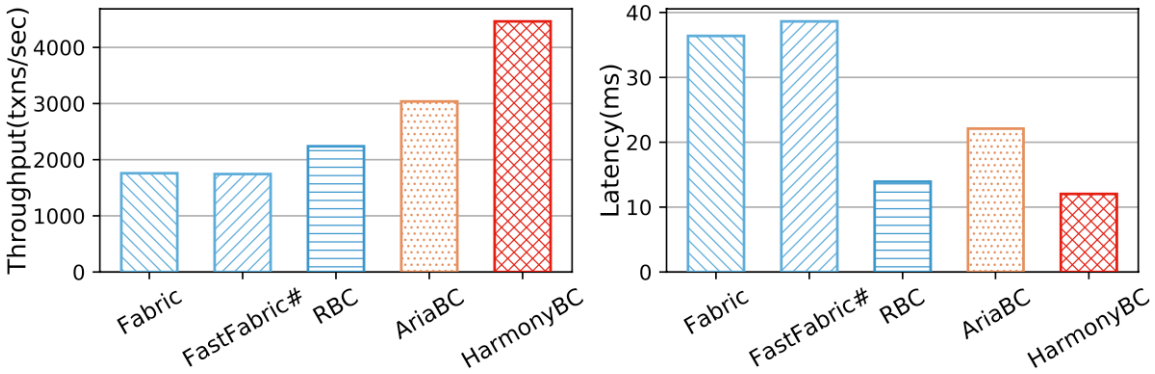
Inter-block Parallelism

	Aria	Harmony
Block i start time	All transaction in block (i-1) finish	When some CPU cycles idle in Block (i-1)
Block i read	Snapshot of block (i-1)	Snapshot of block (i-2)
Commit phase	Patterns A1 and A2	Inter-block dangerous backward structure

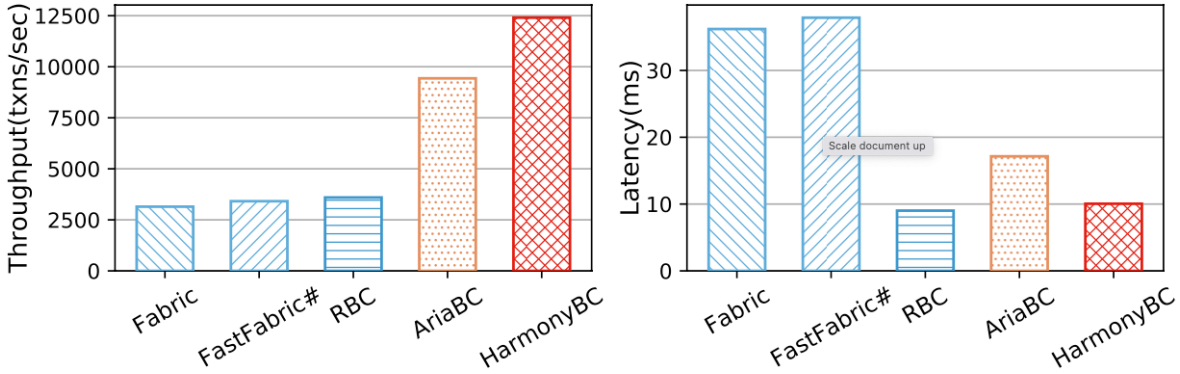
Summary of Harmony

	Aria	Harmony
Design Choice 1	High Abort Rate for Parallel Commit	Low Abort Rate with Parallel Commit
Rationale	Main-memory transactions are short-lived - Hasty abort for easier parallel commit	BC transactions are longer (consensus + disk I/O) - An abort is way more expensive ==> Judicious abort
Design Choice 2	No inter-block parallelism	With inter-block parallelism
Rationale	Not an issue because main-memory transaction lifespans are short and assumed with low variance	- BC transaction lifespans are longer and with higher variance - One straggler transaction in block i-1 would detain subsequent blocks >= I
Design Choice 3	No dealing with hotspot	Deal with hotspot

Empirical Results



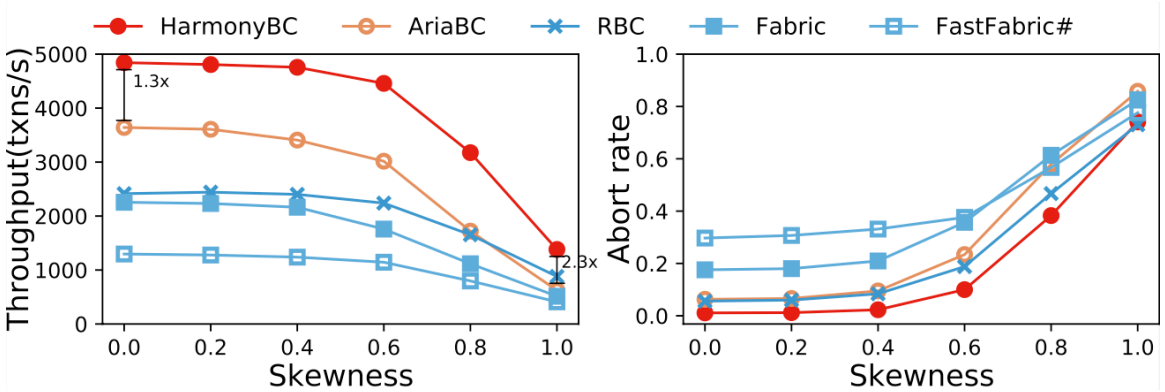
YCSB (5 read and 5 updates per txn, 0.6 zipf theta)



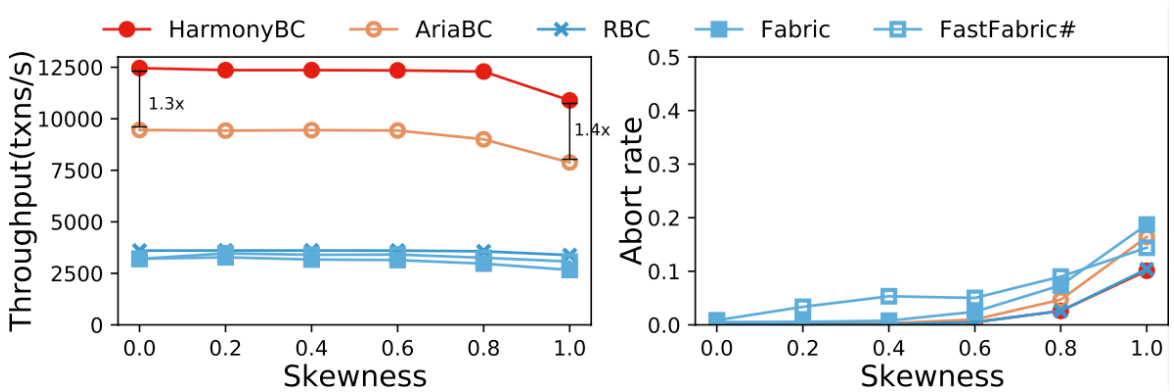
Smallbank (standard mix, 0.6 zipf theta)

- 3.0x – 3.5x throughput over existing blockchains

Empirical Results



YCSB (5 read and 5 updates per txn)



Smallbank (standard mix)

- Harmony is especially better under high contention!

Conclusions

Harmony

- Relational Private Blockchain
- Support full SQL, access control, and recovery using PostgreSQL
- 300%-350% higher throughput than state-of-the-art
- Backing technology:

Deterministic Concurrency Control for Blockchain