Lockless Blockchain Sharding with Multiversion Control

Ramesh Adhikari, Costas Busch

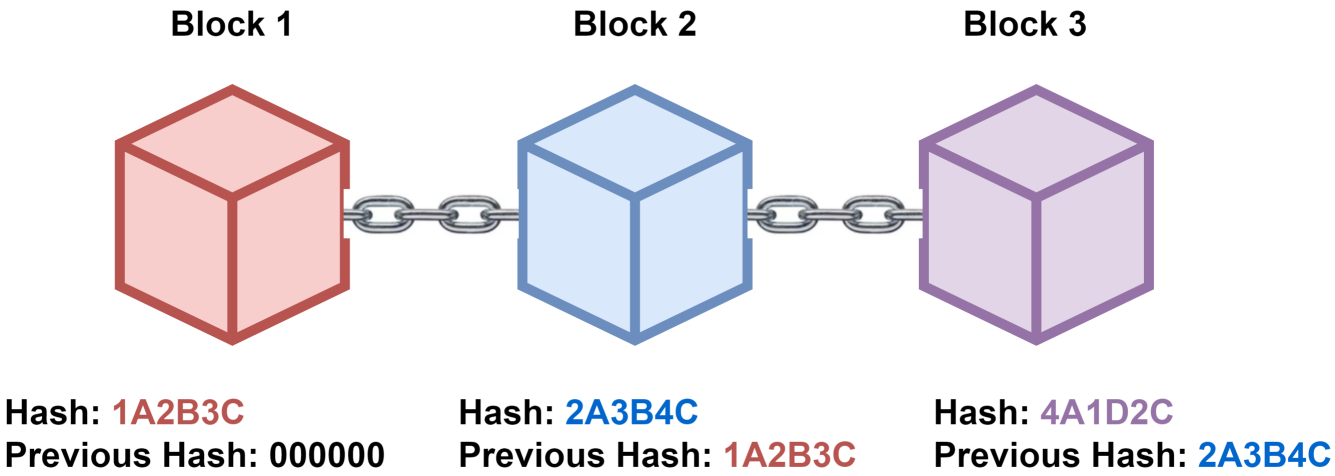
School of Computer and Cyber Sciences
Augusta University
Augusta, Georgia, USA

SIROCCO 2023
June 8th, 2023

Table of Contents

Table of Contents

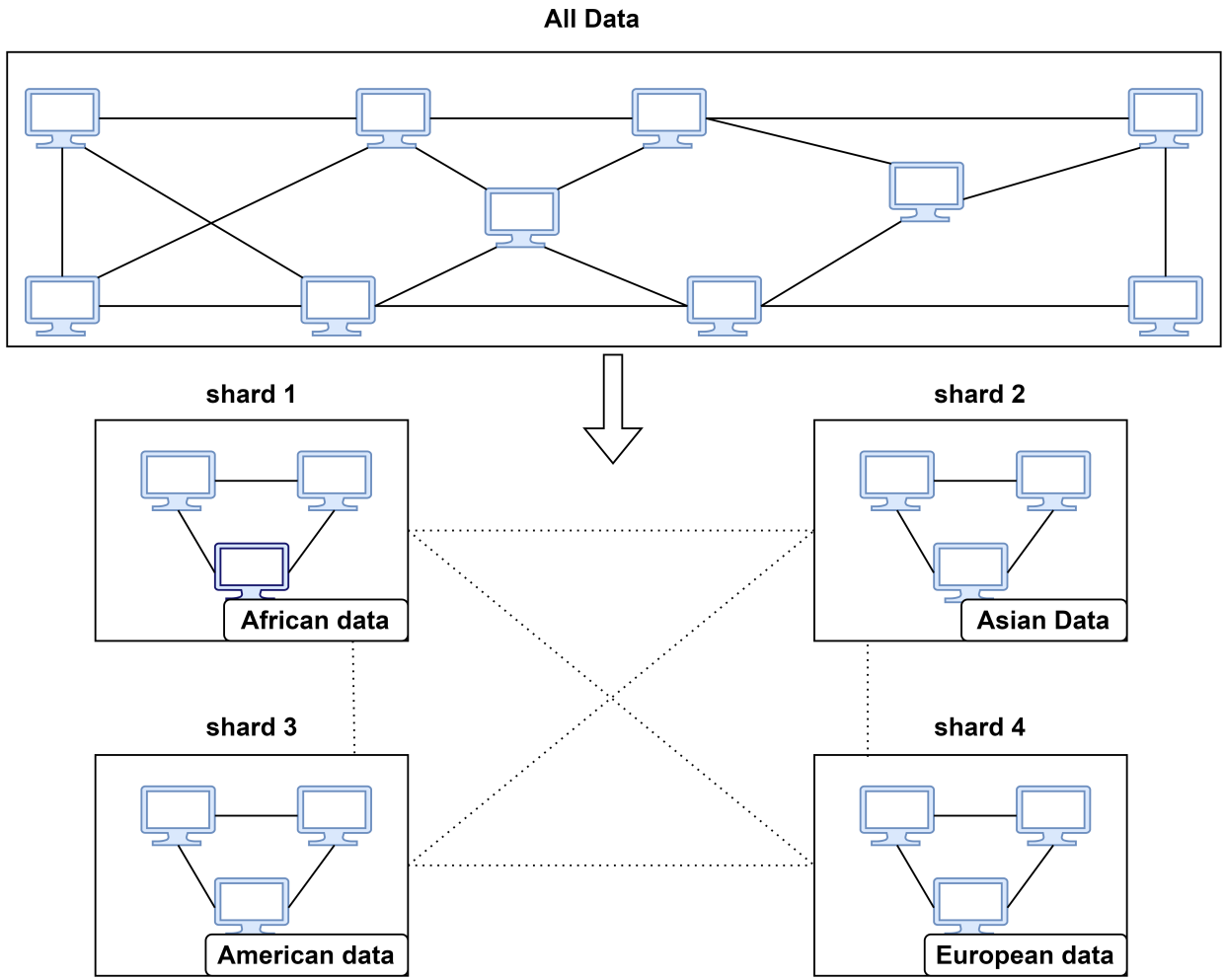
Blockchain and its Scalability Issues



- Blockchain is a chain of blocks that contains a transactions
- It requires all participants to agree on the validity of a transaction
- Increasing the number of nodes slows down transaction propagation and consensus

Solution: Blockchain Sharding

- Sharding splits transaction processing among smaller node groups
- These groups work in parallel to maximize performance



Transactions and Subtransactions for sharding

Example

T_1 = "Transfer 500 from Asma account to Rock account, if Asma has 500 and Rock has 1000".

- Suppose **shard a** holds Asma account and **shard r** holds Rock account
- So we split this transaction into two subtransactions as:
- $T_{1,a}$: "Check Asma has 500"
  : "Remove 500 from Asma acct"
  $T_{1,r}$: "Check Rock has 1000"
  : "Add 500 to Rock acct"

Example: Conflicting Transactions

Example

T_1 = "Transfer 500 from Asma account to Rock account, if Asma has 500 and Rock has 1000"

T_2 = "Transfer 200 from Rock account to Asma account, if Asma has 100 and Rock has 200"

$T_{1,a}$: "Check Asma has 500"
: "Remove 500 from Asma acct"

$T_{1,r}$: "Check Rock has 1000"
: "Add 500 to Rock acct"

$T_{2,a}$: "Check Asma has 100"
: "Add 200 to Asma acct"

$T_{2,r}$: "Check Rock has 200"
: "Remove 200 from Rock acct"

Example: Sharding with conflicting Transactions

Example

T_1 = "Transfer 500 from Asma account to Rock account, if Asma has 500 and Rock has 1000"

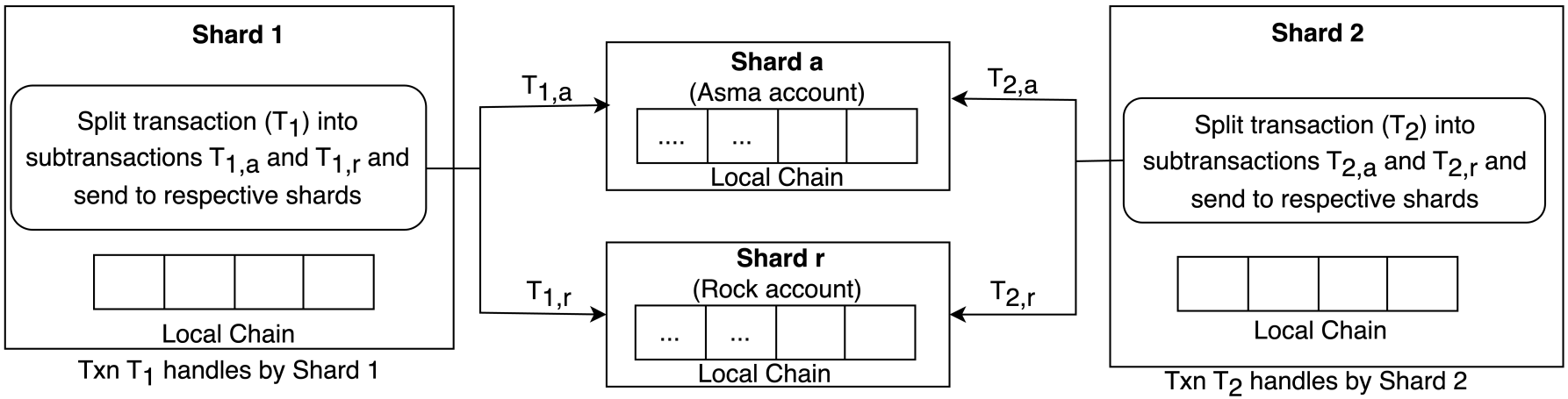
T_2 = "Transfer 200 from Rock account to Asma account, if Asma has 100 and Rock has 200"

$T_{1,a}$: "Check Asma has 500"
: "Remove 500 from Asma acct"

$T_{1,r}$: "Check Rock has 1000"
: "Add 500 to Rock acct"

$T_{2,a}$: "Check Asma has 100"
: "Add 200 to Asma acct"

$T_{2,r}$: "Check Rock has 200"
: "Remove 200 from Rock acct"



Example: Sharding with conflicting Transactions

Example

T_1 = "Transfer 500 from Asma account to Rock account, if Asma has 500 and Rock has 1000"

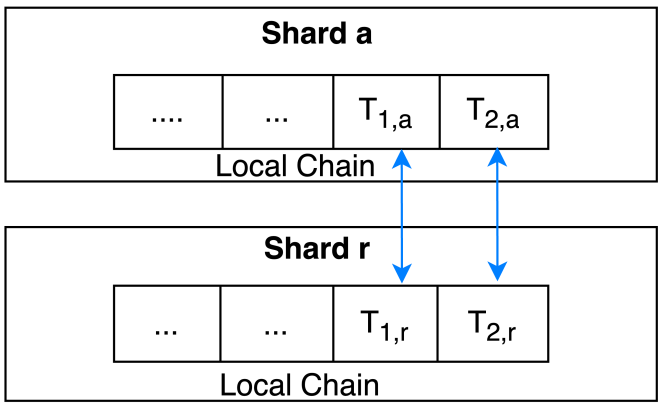
T_2 = "Transfer 200 from Rock account to Asma account, if Asma has 100 and Rock has 200"

$T_{1,a}$: "Check Asma has 500"
: "Remove 500 from Asma acct"

$T_{1,r}$: "Check Rock has 1000"
: "Add 500 to Rock acct"

$T_{2,a}$: "Check Asma has 100"
: "Add 200 to Asma acct"

$T_{2,r}$: "Check Rock has 200"
: "Remove 200 from Rock acct"



correct order of local commit in **Shard a** and **Shard r** can serialize transactions later

Serialization issue in sharding

Example

T_1 = "Transfer 500 from Asma account to Rock account, if Asma has 500 and Rock has 1000"

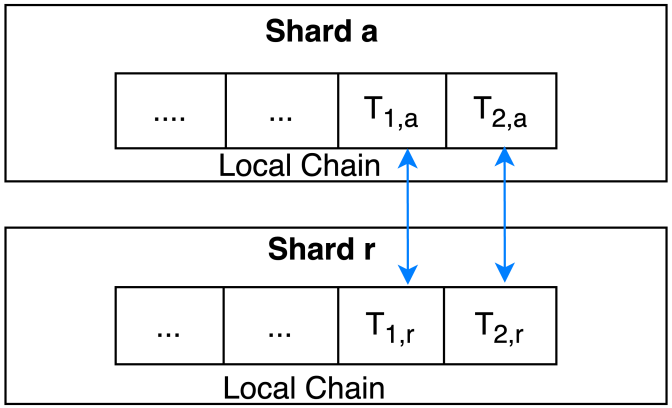
T_2 = "Transfer 200 from Rock account to Asma account, if Asma has 100 and Rock has 200"

$T_{1,a}$: "Check Asma has 500"
: "Remove 500 from Asma acct"

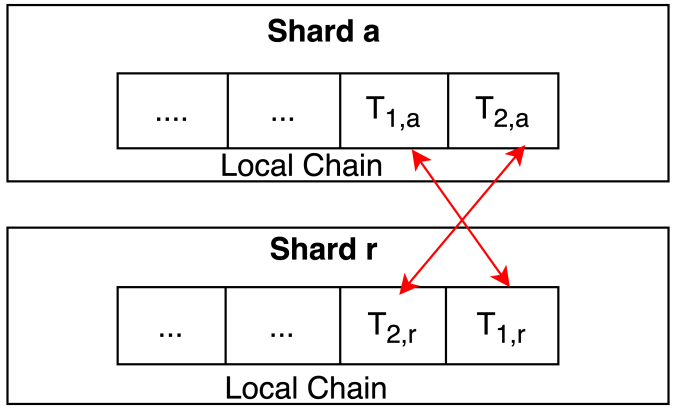
$T_{1,r}$: "Check Rock has 1000"
: "Add 500 to Rock acct"

$T_{2,a}$: "Check Asma has 100"
: "Add 200 to Asma acct"

$T_{2,r}$: "Check Rock has 200"
: "Remove 200 from Rock acct"



correct order of local commit in **Shard a** and **Shard r** can serialize transactions later



Incorrect order of local commit in **Shard a** and **Shard r** cannot serialize transactions later

Motivation and Related Work

- To use sharding, we need some kind of synchronization mechanisms to commit transactions and subtransactions safely in different shards
- However, existing sharding solutions like Byshard [1], OmniLedger [2] use locks for transaction isolation and concurrency control

Motivation and Related Work

- To use sharding, we need some kind of synchronization mechanisms to commit transactions and subtransactions safely in different shards
- However, existing sharding solutions like Byshard [1], OmniLedger [2] use locks for transaction isolation and concurrency control
- Lock lowers the system throughput

Motivation and Related Work

- To use sharding, we need some kind of synchronization mechanisms to commit transactions and subtransactions safely in different shards
- However, existing sharding solutions like Byshard [1], OmniLedger [2] use locks for transaction isolation and concurrency control
- Lock lowers the system throughput
- Hagar Meir et al. [3] present a lock-free method for transaction isolation, but they do not address a sharding-based blockchain model

Our Contributions

- We provide a lockless protocol for sharded blockchains
  - Instead of using locks, we use a multi-version approach for concurrency control
  - If conflicts occur between two transactions in any shard, one of them will restart

Our Contributions

- We provide a lockless protocol for sharded blockchains
  - Instead of using locks, we use a multi-version approach for concurrency control
  - If conflicts occur between two transactions in any shard, one of them will restart
- We provide formal correctness proofs for the safety and liveness of our proposed protocol

Our Contributions

- We provide a lockless protocol for sharded blockchains
  - Instead of using locks, we use a multi-version approach for concurrency control
  - If conflicts occur between two transactions in any shard, one of them will restart
- We provide formal correctness proofs for the safety and liveness of our proposed protocol
- We also provide an experimental analysis of our protocol

Table of Contents

Our Sharding Model

- Shard that initiates and handles the processing of the transaction called **Leader Shard**
- Shards that verify received subtransactions and append valid ones to their local ledgers, called **Destination Shard**

Our Sharding Model

- Shard that initiates and handles the processing of the transaction called **Leader Shard**
- Shards that verify received subtransactions and append valid ones to their local ledgers, called **Destination Shard**
- Each shard only contains the local chain

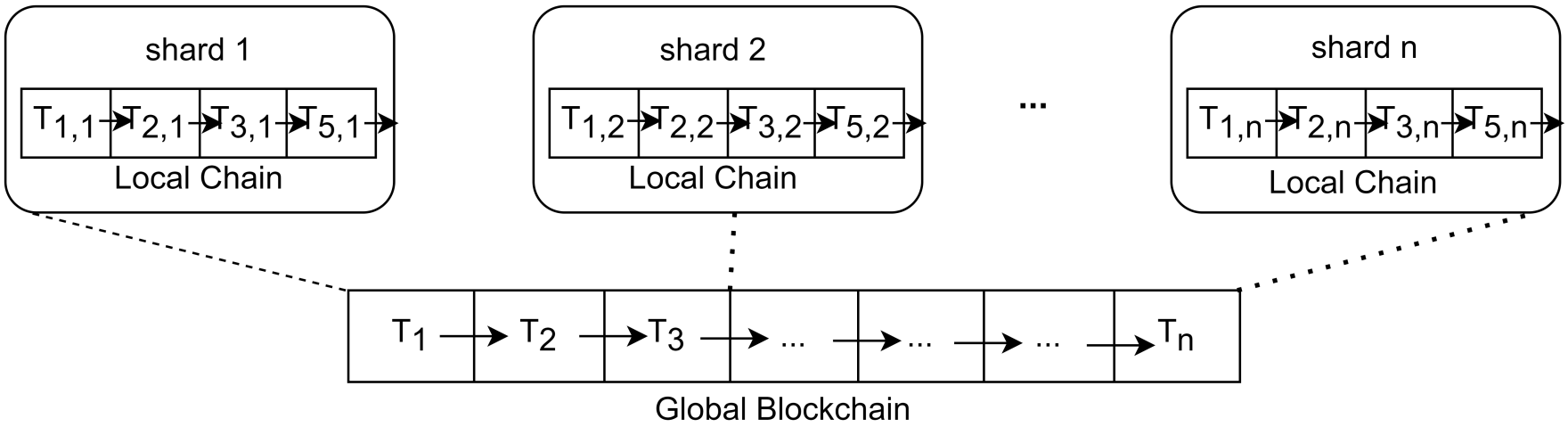


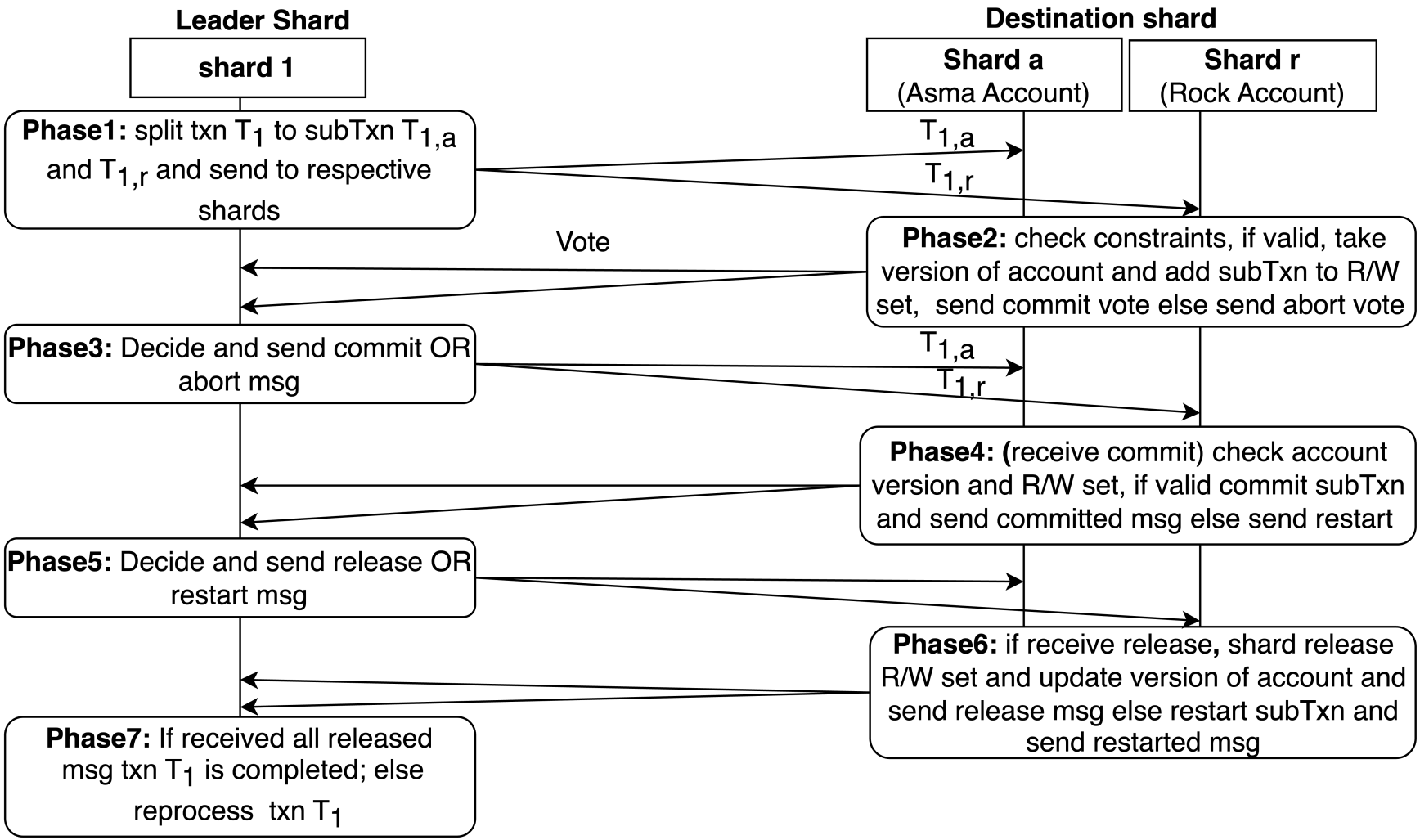
Table of Contents

Sharding Algorithm

T_1 = Transfer 500 from Asma Account to Rock account, if Asma has 500 and Rock has 1000

$T_{1,a}$: "Check Asma has 500" & "Remove 500 from Asma account"

$T_{1,r}$: "Check Rock has 1000" & "Add 500 to Rock account"



Our protocol provides Safety

If two transactions conflict in any shard, one of them will restart

We achieve Safety by using **Write set** and **Account Version**

Our protocol provides Safety

If two transactions conflict in any shard, one of them will restart

We achieve Safety by using **Write set** and **Account Version**

- Suppose two subtransactions $T_{1,a}$ and $T_{2,a}$ conflict in **shard a** by accessing the **Asma** account
- Let $T_{1,a}$ has finished executing phase 2. Hence, $\mathbf{T_{1,a}}$ **has been added to write set W(Asma)**

Our protocol provides Safety

If two transactions conflict in any shard, one of them will restart

We achieve Safety by using **Write set** and **Account Version**

- Suppose two subtransactions $T_{1,a}$ and $T_{2,a}$ conflict in **shard a** by accessing the **Asma** account
- Let $T_{1,a}$ has finished executing phase 2. Hence, $\mathbf{T_{1,a}}$ **has been added to write set W(Asma)**
- Later, when $T_{2,a}$ reaches phase 4 (i.e Commit phase)
  - It will observe $\mathbf{T_{1,a}}$ **is already in W(Asma)** which will force $T_{2,a}$ to restart
  - On the other hand, if T_1 completed and updated the version of Asma. Then $T_{2,a}$ will restart by observing the updated version of Asma
- In either case, one of the two transactions will restart

Our protocol provides Liveness

Our protocol guarantees that every transaction will eventually commit

Our protocol provides liveness by **prioritizing transaction** execution

- Periodically, each shard sends its lowest transaction ID to other shards
- In case of conflict, priority is given to lowest ID

Our protocol provides Liveness

Our protocol guarantees that every transaction will eventually commit

Our protocol provides liveness by **prioritizing transaction** execution

- Periodically, each shard sends its lowest transaction ID to other shards
- In case of conflict, priority is given to lowest ID
- Suppose two subtransactions $T_{1,a}$ and $T_{2,a}$ conflict in **shard a**
- Assume T_1 has highest priority

Our protocol provides Liveness

Our protocol guarantees that every transaction will eventually commit

Our protocol provides liveness by **prioritizing transaction** execution

- Periodically, each shard sends its lowest transaction ID to other shards
- In case of conflict, priority is given to lowest ID
- Suppose two subtransactions $T_{1,a}$ and $T_{2,a}$ conflict in **shard a**
- Assume T_1 has highest priority
- Suppose subtransaction $T_{2,a}$ is in between the phase 2 and 6
- Then when $T_{1,a}$ reaches phase 4. It will observe inprocessing of $T_{2,a}$. As $T_{1,a}$ has the highest priority to execute which will force $T_{2,a}$ to rollback while $T_{1,a}$ make progress

Table of Contents

Experimental Analysis

- Transaction execution time of protocol is better than the lock-based protocol

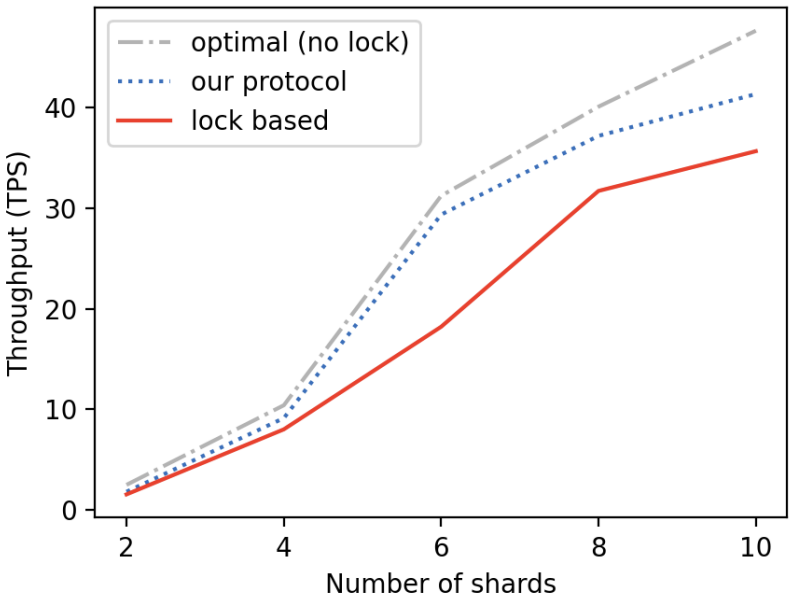


Figure: (1). Average transaction throughput with the number of shards

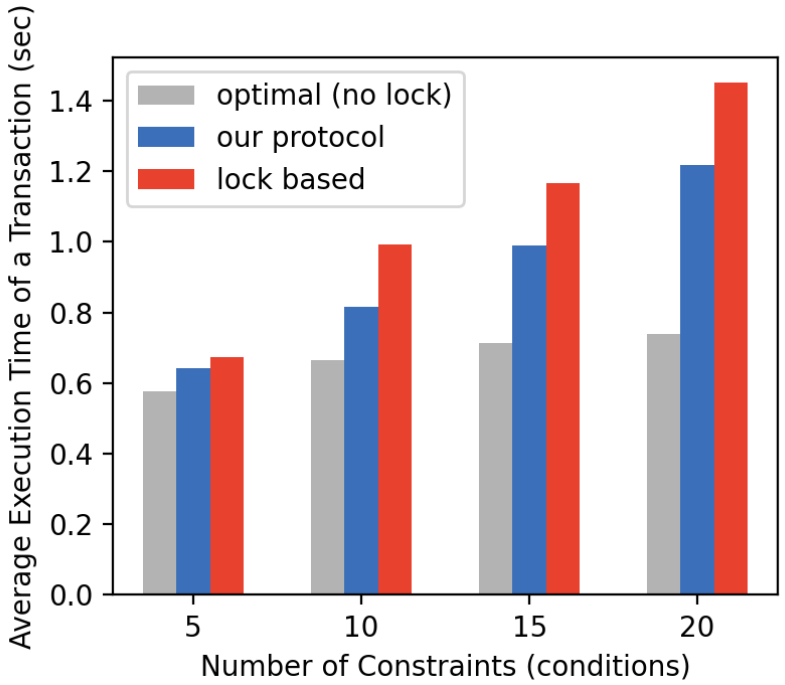


Figure: (2). Average execution time of a transaction with numbers of constraints

Table of Contents

Conclusion future work

- We presented a lockless transaction protocol for blockchain sharding
- Our protocol relies on taking versions of the shared objects
- Proposed protocol achieves faster transaction execution times compared to the lock-based approach

Conclusion future work

- We presented a lockless transaction protocol for blockchain sharding
- Our protocol relies on taking versions of the shared objects
- Proposed protocol achieves faster transaction execution times compared to the lock-based approach
- Future work: Explore efficient communication between shards

References

- Hellings, J., Sadoghi, M.: Byshard: Sharding in a byzantine environment. Proceedings of the VLDB Endowment **14**(11), 2230–2243 (2021)
- Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger via sharding. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 583–598. IEEE (2018)
- Meir, H., Barger, A., Manevich, Y., Tock, Y.: Lockless transaction isolation in hyperledger fabric. In: 2019 IEEE International Conference on Blockchain (Blockchain). pp. 59–66 (2019). https://doi.org/10.1109/Blockchain.2019.00017

Thank you!