DAG-based Efficient Parallel Scheduler for Blockchains: Hyperledger Sawtooth as a Case Study

Manaswini Piduguralla   Saheli Chakraborty   Parwat Singh Anjana
Sathya Peri

Euro-Par 2023

01 September 2023

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Table of Contents

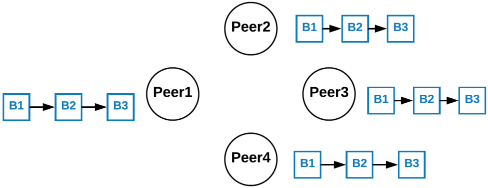
Table of Contents

Research Objective

- The objective is to develop an Efficient Distributed and Secure framework for the Execution of Smart Contracts in Blockchains.
  - Develop framework for concurrent execution of transactions.

Blockchain

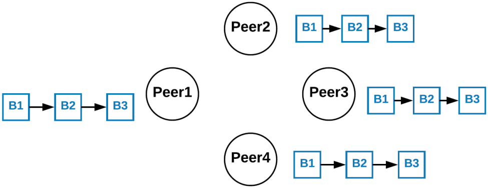
- Blockchain is a decentralized distributed immutable ledger shared among untrusted parties[1].

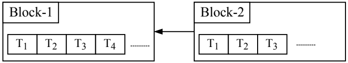


[1] **Nakamoto:Bitcoin:2009**.

Blockchain

- Blockchain is a decentralized distributed immutable ledger shared among untrusted parties[1].





- Each block contains a set of transactions.

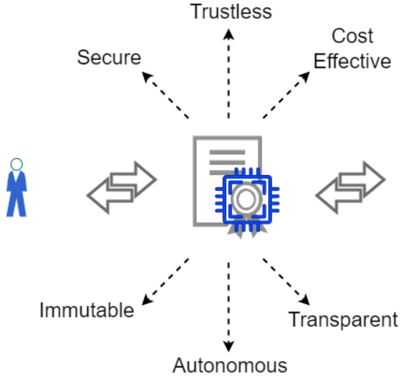
[1] **Nakamoto:Bitcoin:2009**.

Smart Contracts

- Smart contracts (SCs) are self executing contracts with agreement between two or more parties that are written in the form of computer code.
- Smart contracts are like a ‘class’ that encapsulates data and methods.

Smart Contracts

- Smart contracts (SCs) are self executing contracts with agreement between two or more parties that are written in the form of computer code.
- Smart contracts are like a ‘class’ that encapsulates data and methods.



- Blockchain nodes form a peer-to-peer system.
- Clients (external to the system) wishing to execute transactions, contact a peer of the system.

[2] **Parwat+:Netys:2020**.

Introduction: Blockchain High Level Design

- Blockchain nodes form a peer-to-peer system.
- Clients (external to the system) wishing to execute transactions, contact a peer of the system.

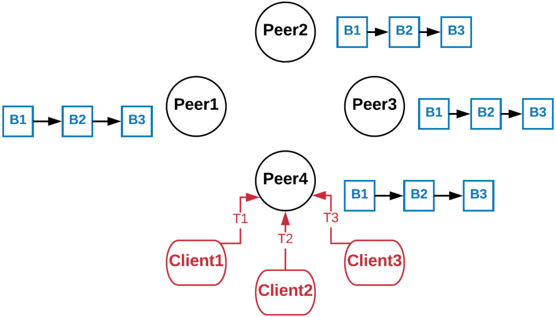


Figure: Clients send Transaction T1, T2 and T3 to block producer (Peer4)[2]

[2] **Parwat+:Netys:2020**.

Introduction: Blockchain High Level Design

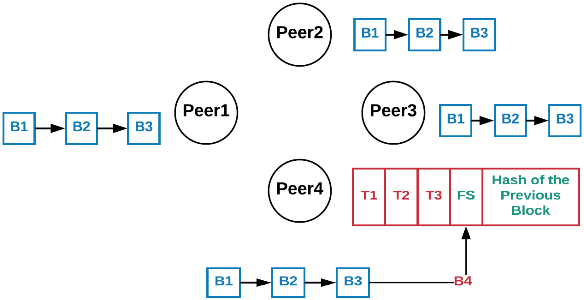


Figure: Block producer forms a block B4 and computes final state (FS) sequentially

Introduction: Blockchain High Level Design

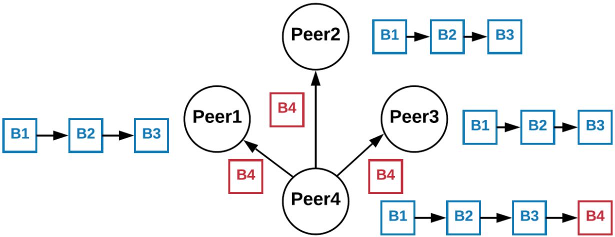


Figure: Block producer broadcasts the block B4

Introduction: Blockchain High Level Design

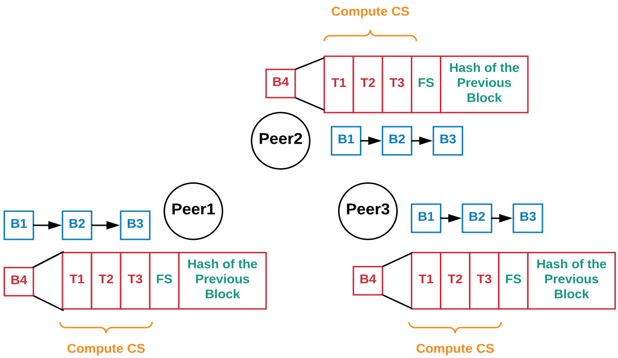


Figure: Validators (Peer 1, 2, and 3) compute current state (CS) sequentially

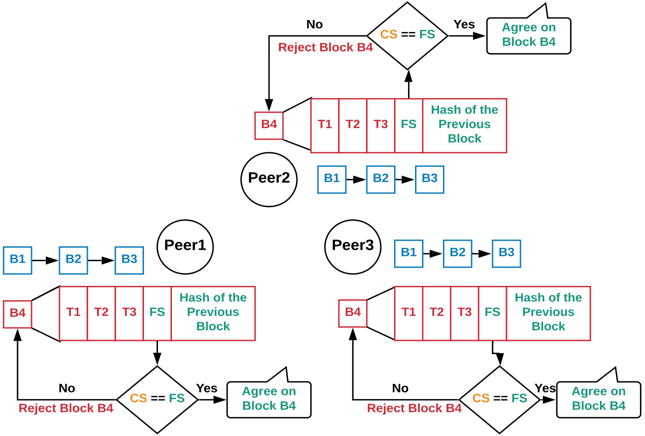


Figure: Validators verify the FS and reach the consensus protocol

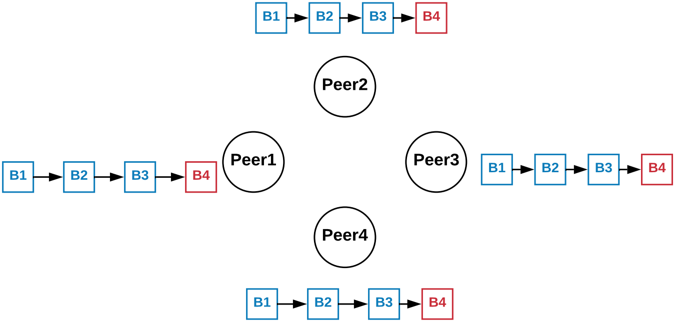


Figure: Block B4 successfully added to the blockchain

Table of Contents

- **Hyperledger Sawtooth**[3] is a modular platform for building, deploying, and running distributed ledgers.
- The modular property of sawtooth, enables enterprises and consortiums to make decisions about their blockchain applications for themselves.

[3] **sawtooth:url**.

- **Hyperledger Sawtooth**[3] is a modular platform for building, deploying, and running distributed ledgers.
- The modular property of sawtooth, enables enterprises and consortiums to make decisions about their blockchain applications for themselves.

Sawtooth Key Features:

- Modular
- Permissioned as well as permissionless infrastructure
- Parallel transaction execution
- Pluggable consensus algorithms
- Multi language support
- Dynamic consensus

Sawtooth

[3] **sawtooth:url**.

Table of Contents

Motivation

Smart contract in blockchain are executed in two different contexts :

Smart contract in blockchain are executed in two different contexts :

- While block producers are composing the blocks.

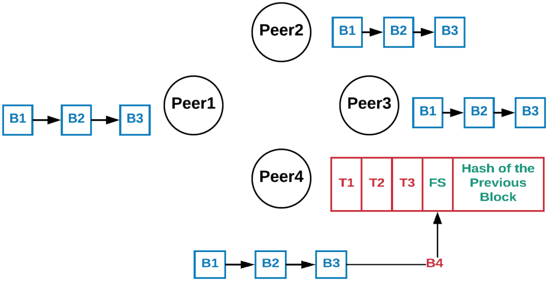


Figure: Block producer create the block and broadcast it to others

Motivation

Smart contract in blockchain are executed in two different contexts :

- While block producers are composing the blocks.
- While nodes/peers are validating the blocks.

Motivation

Smart contract in blockchain are executed in two different contexts :

- While block producers are composing the blocks.
- While nodes/peers are validating the blocks.

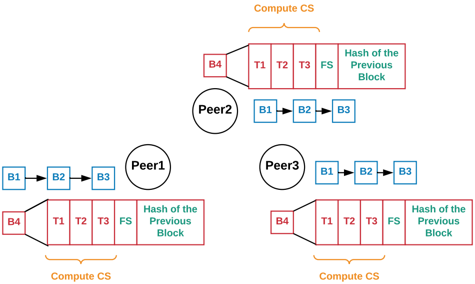


Figure: Validators (Peer 1, 2, and 3) compute current state (CS) sequentially

In both stages, the transactions in the block are executed serially in most blockchains[4]:

- Not utilizing the multi-core processors efficiently.
- Results in lower throughput.

[4] **Dickerson+:ACSC:PODC:2017**.

In both stages, the transactions in the block are executed serially in most blockchains[4]:

- Not utilizing the multi-core processors efficiently.
- Results in lower throughput.

Solution: *Concurrent execution of SCTs*

[4] **Dickerson+:ACSC:PODC:2017**.

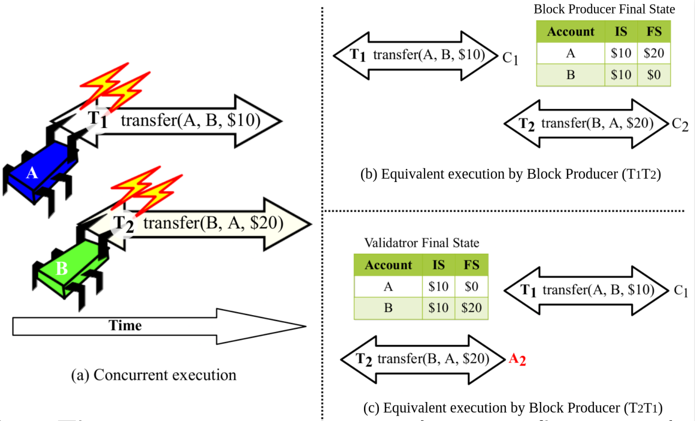
- Validator may incorrectly reject a valid block proposed by the block producer. We call such error as **False Block Rejection (FBR)** error[5],[6].

[5] **Dickerson+:ACSC:PODC:2017**.

[6] **Parwat+:Netys:2020**.

Concurrent Execution Challenges (1/3)

- Validator may incorrectly reject a valid block proposed by the block producer. We call such error as **False Block Rejection (FBR)** error.



Solution: The concurrent execution is always conflict equivalent to a serial schedule.

Contributions of this work

- We have proposed a concurrent transaction execution framework for blockchains.
- The proposed approach has been thoroughly tested in Hyperledger Sawtooth 1.2.6.
- It is flexible enough for implementation in any blockchain that follows the order-execute paradigm.

Table of Contents

Framework Design

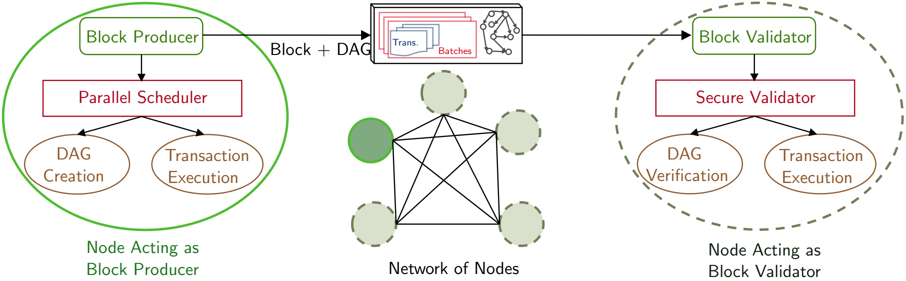


Figure: Proposed framework in the blockchain

Proposed Parallel Scheduler using DAG

- The parallel scheduler performs the operations of DAG creation and conflict-free transaction execution.

Proposed Parallel Scheduler using DAG

- The parallel scheduler performs the operations of DAG creation and conflict-free transaction execution.
  - DAG (Direct Acyclic Graph) to represent the dependencies among the transactions:
    - The nodes of the graph will be the transactions in the block.

Proposed Parallel Scheduler using DAG

- The parallel scheduler performs the operations of DAG creation and conflict-free transaction execution.

  - DAG (Direct Acyclic Graph) to represent the dependencies among the transactions:
    - The nodes of the graph will be the transactions in the block.
    - An edge represents the dependency between two transactions.

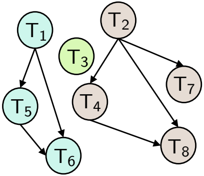


Figure: DAG representation of dependencies in the block.

- Dependency:
  When two transactions are accessing the same data while at least one of which is modifying it.
  - T1 : View(A)
  - T5: Send(B, A, 10)

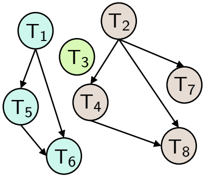


Figure: DAG representation of dependencies in the block.

Proposed Parallel Scheduler using DAG

- Scheduling:
  - Transactions with zero indegree are not dependent on current set of executing transactions.
  - These transactions can be scheduled for execution.

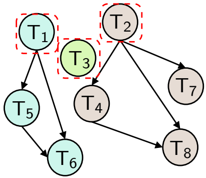


Figure: DAG representation of dependencies in the block.

Proposed Parallel Scheduler using DAG

- Scheduling:
  - The out edges are removed of the transactions that have completed execution.

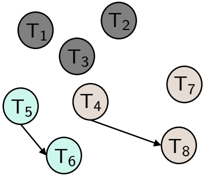


Figure: DAG representation of dependencies in the block.

Framework Design

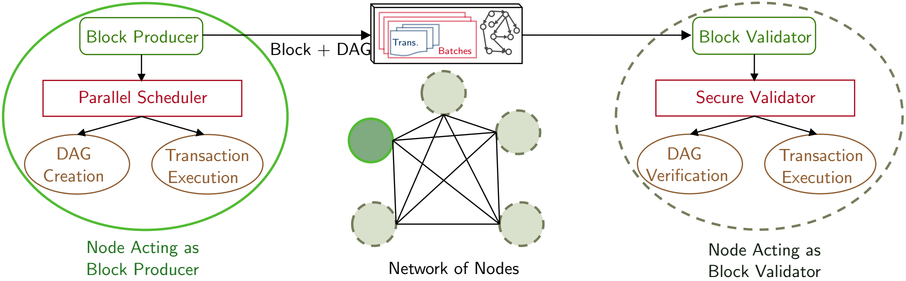


Figure: Proposed framework in the blockchain

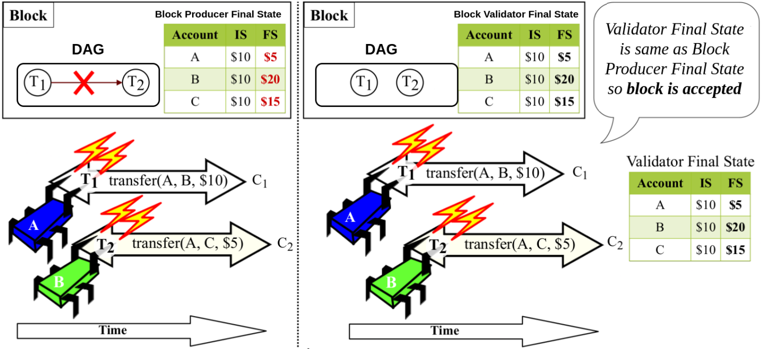
Concurrent Execution Challenges (2/3)

- The *Malicious block producer* can send an incorrect Block Graph to harm the blockchain, missing some edges, to cause *double spending*. We call such error as **Edge Missing BG (EMB)** error[7].

[7] **Parwat+:Netys:2020**.

Concurrent Execution Challenges (2/3)

- The *Malicious block producer* can send an incorrect Block Graph to harm the blockchain, missing some edges, to cause *double spending*. We call such error as **Edge Missing BG (EMB)** error.



Block Producer Final State

Account	IS	FS
A	$10	$5
B	$10	$20
C	$10	$15

Block Validator Final State

Account	IS	FS
A	$10	$5
B	$10	$20
C	$10	$15

Validator Final State

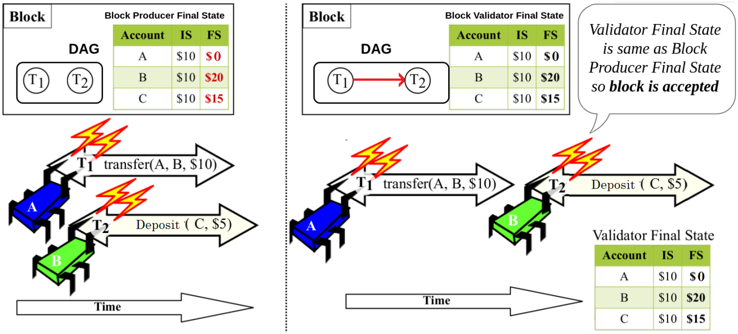
Account	IS	FS
A	$10	$5
B	$10	$20
C	$10	$15

Solution: We propose a *Secure Multi-threaded Validator (SMV)* to detect EMB error and rejects the corresponding blocks.

Concurrent Execution Challenges (3/3)

- The *Malicious block producer* can send an incorrect Block Graph with extra edges, to intentionally slow the validation process. We call such error as **Extra Edge BG (EEB)** error.

- The *Malicious block producer* can send an incorrect Block Graph with extra edges, to intentionally slow the validation process. We call such error as **Extra Edge BG (EEB)** error.



Solution: We propose a *Secure Multi-threaded Validator (SMV)* to detect EEB error and rejects the corresponding blocks.

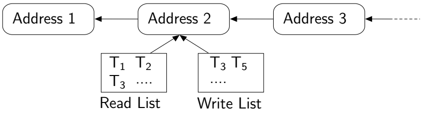


Figure: Linked list address data for secure validator

- **Edge Validation:** There should be an edge connecting each transaction in the read list to every transaction in the write list.
- An edge should also connect any two transactions within the write list.

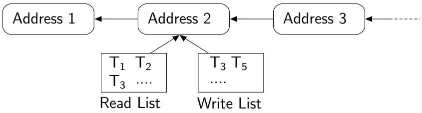
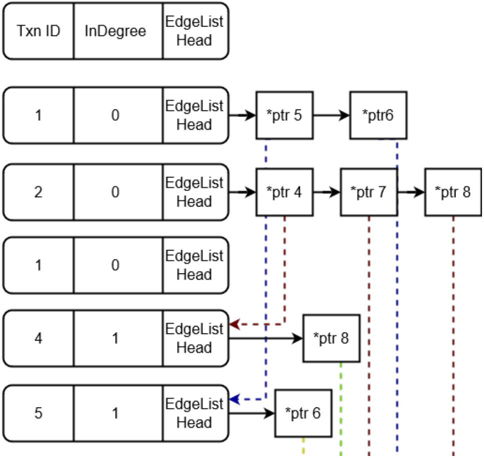


Figure: Linked list address data for secure validator

- **Edge Validation:** There should be an edge connecting each transaction in the read list to every transaction in the write list.
- An edge should also connect any two transactions within the write list.
- **In-degree validation:** We track the in-degree of each transaction during edge validation and cross check it with DAG in-degree.

DAG Implementations

- We have developed two implementations for the proposed DAG scheduler:
  - Linked List
  - Adjacency Matrix



Transaction Families

We implemented four transaction families to test the performance of our approach:

Transaction Families

We implemented four transaction families to test the performance of our approach:

- IntKey Transaction Family
- SmallBank Transaction Family
- Voting Transaction Family
- Insurance Transaction Family
- Mixed block containing all three

Table of Contents

- Voting Transaction Family
  - Y-axis: Time in seconds
  - X-axis: Number of blocks
  - Number of threads: 56
  - Degree of dependency: 50% CP3
  - Number of txns: 1000

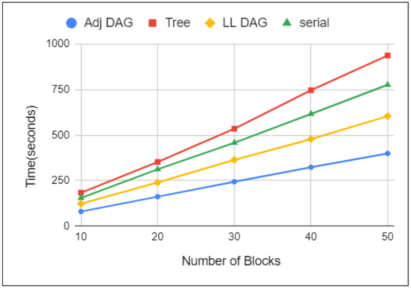


Figure: Experiment one: Voting

Curves: Serial Scheduler, Tree Parallel Scheduler, LLDAG Scheduler, Adj DAG Scheduler.

- SimpleWallet Transaction Family
  - Y-axis: Time in seconds
  - X-axis: Number of transactions per block
  - Number of threads: 56
  - Degree of dependency: 50% CP2
  - Number of blocks: 20

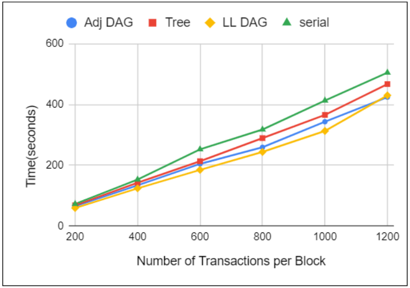


Figure: Experiment two: SimpleWallet

Curves: Serial Scheduler, Tree Parallel Scheduler, LLDAG Scheduler, Adj DAG Scheduler.

- SimpleWallet Transaction Family
  - Y-axis: Time in seconds
  - X-axis: Number of transactions per block
  - Number of threads: 56
  - Number of blocks: 20
  - Number of txns: 1000

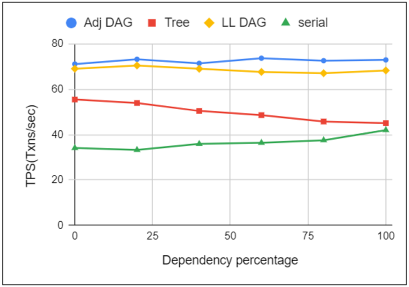


Figure: Experiment three: Mixed Block

Curves: Serial Scheduler, Tree Parallel Scheduler, LLDAG Scheduler, Adj DAG Scheduler.

- Dependency data structure creation or validation time for SimpleWallet Transaction Family
  - Y-axis: Time in seconds
  - X-axis: Number of transactions per block
  - Number of threads: 56
  - Degree of dependency: 50% CP2
  - Number of blocks: 20

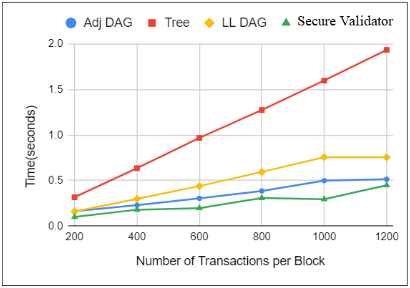


Figure: Comparison of data structure creation time.

Curves: Tree Parallel Scheduler, LLDAG Scheduler, Adj DAG Scheduler, Secure Validator

Table of Contents

Conclusion

- We have introduced a concurrent transaction execution framework for blockchain systems.
- Rigorous testing of our approach has taken place within Hyperledger Sawtooth 1.2.6.

[8] **Piduguralla:artifact:2023**.

Conclusion

- We have introduced a concurrent transaction execution framework for blockchain systems.
- Rigorous testing of our approach has taken place within Hyperledger Sawtooth 1.2.6.
- Our adaptable framework is compatible with any blockchain adhering to the order-execute paradigm.
- The artifact of our framework is evaluated and is available in Figshare repository[8].

[8] **Piduguralla:artifact:2023**.

Future Work

- Optimize the framework by improving DAG creation and secure validation is our immediate next step.
- We will be exploring enhancing the fault tolerance and scalability for each blockchain node.

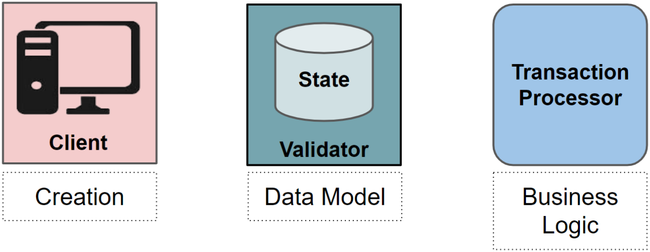
Thank you

Thank you to ACM-W and Google for sponsoring my attendance and paper presentation at this conference

Extras

Transaction Families

Transaction Families ≈ Smart Contracts



Intro

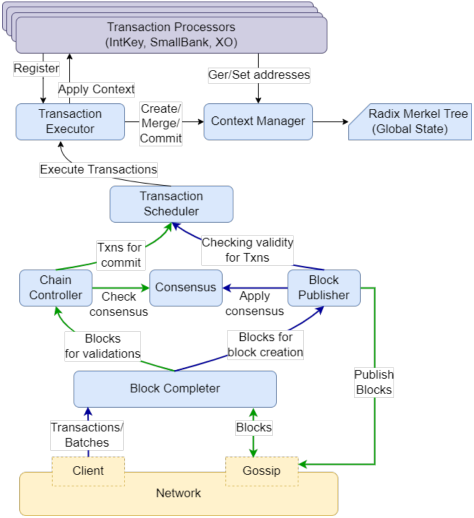
Consensus

- The **journal** is the group of validator subcomponents that work together to handle batches and proposed blocks.
- The **Block Completer** initially receives the blocks and batches. It guarantees that all dependencies for the blocks have been satisfied.
- Completed batches go to the **Block Publisher** for batch validation and inclusion in a block.
- Completed blocks go to the **Block Validator** for validation and fork resolution.

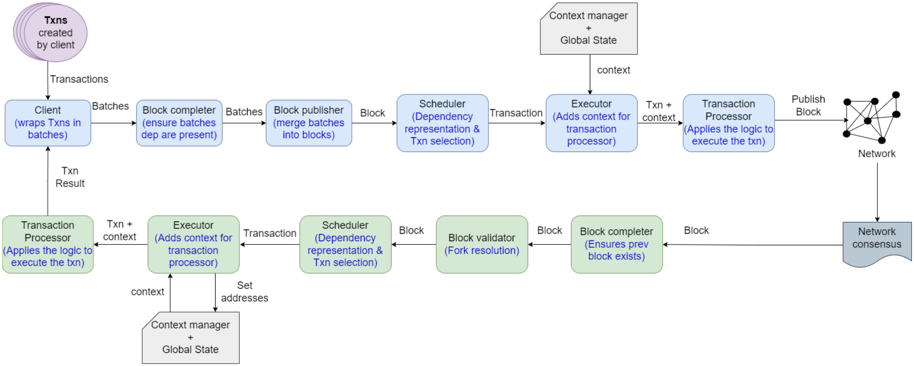
Intro

Sawtooth architecture

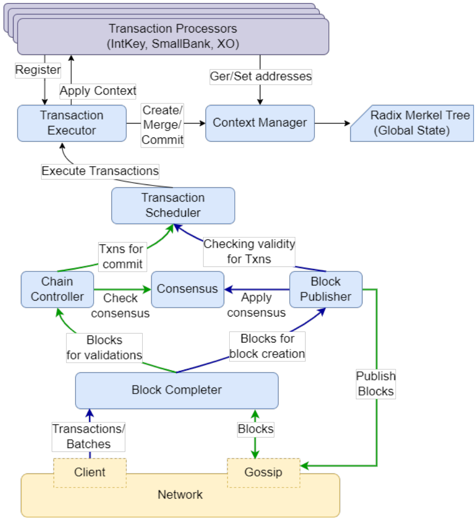
- Journal
- Transaction Scheduler
- Transaction Executor
- Transaction Processor
- Global State



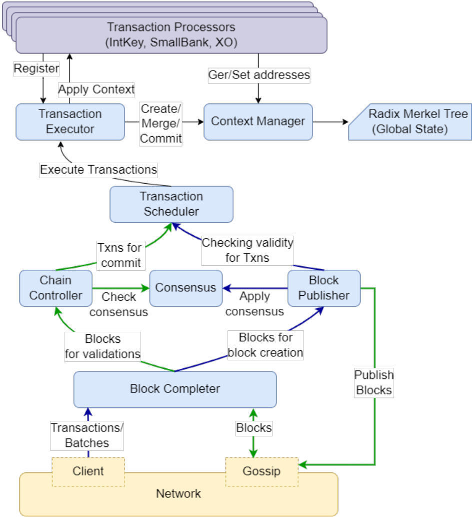
Life Cycle of a Sawtooth Transaction



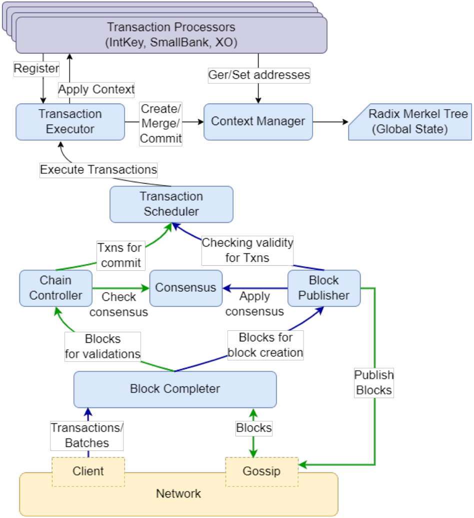
- **Block Completer:** Receives the blocks and batches from the network



- **Block Publisher:** Creation of blocks after validation

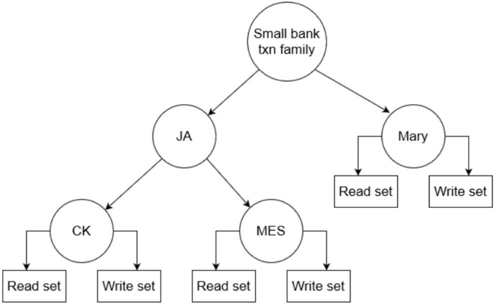


- **Block Validator:** Performs validation and fork resolution for transactions in the block



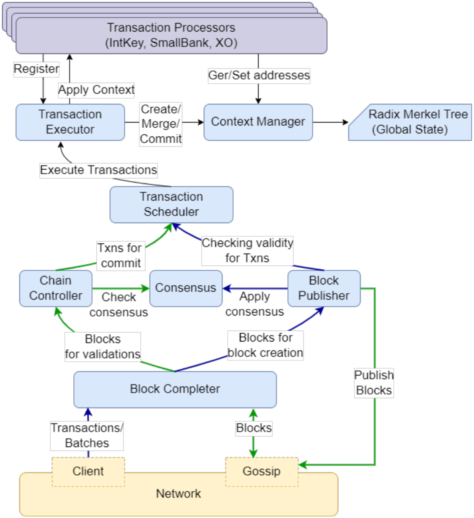
Sawtooth Parallel Scheduler

- Sawtooth tree structure to keep track of dependencies using merkle tree.
- Each data location has read and write lists that contain the ID of the transactions that are accessing them.
- David, Daniel, Dylan.



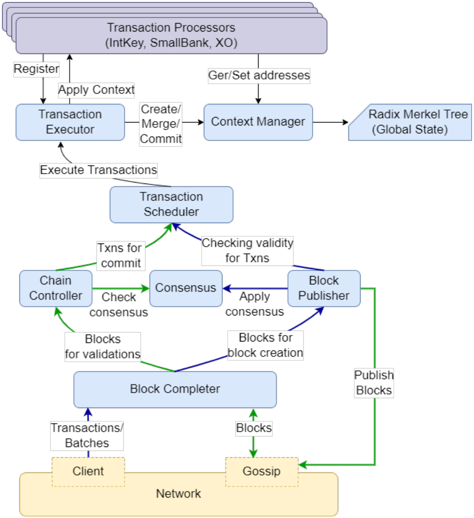
Transaction Executor

- The executor obtains the next transaction from the sceduler.
- Acquires context reference from context manager and state.
- Combines transaction and context for transaction processor.
- The Executor updates the sceduler with the transactions's result with the updated context using locks.



Sawtooth Global State

- Sawtooth uses an addressable Radix Merkle tree to store data for transaction families.
- Merkle tree : stores successive node hashes from leaf-to-root.
- Radix tree: addresses uniquely identify the paths to leaf nodes.
- An address is a hex-encoded string of 35 bytes, each node has 2^8 possible children.



Key places for optimization in Hyperledger Sawtooth

- Sawtooth architecture uses tree with addresses as nodes to represent the dependencies.

$$(Total\ number\ of\ addresses \gg Transactions\ in\ a\ block)$$

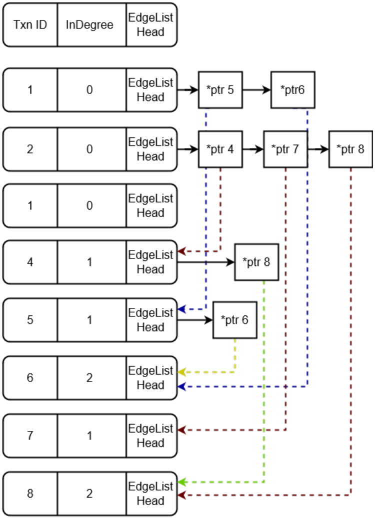
- The construction of the tree is done serially.
- Status of all the dependent transactions is checked before scheduling a transaction.

$$(This\ generates\ a\ delay\ in\ scheduling\ the\ transaction)$$

- The available transaction are searched from the start of the array each time a transaction is scheduled.
- Locks are used to update transaction status.

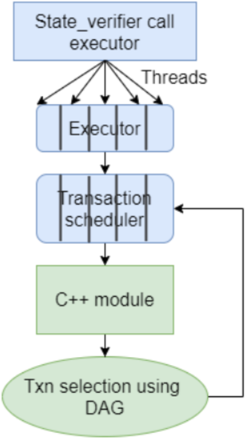
DAG Implementations

- We have developed two implementations for the proposed DAG scheduler:
  - Linked List
  - Ajacency Matrix



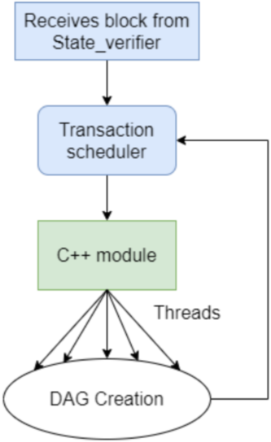
DAG Implementations

- For DAG scheduling the function "Next_transaction" is modified.
- Transactions are selected for scheduling using the DAG created.
- In selecting a transaction there are multiple C++ modules initiated by python threads.



DAG Implementations

- For DAG creation we have modified the "add_batch" function in Sawtooth scheduler.
- The creation of DAG is done through multiple threads in C++ module.



Validators

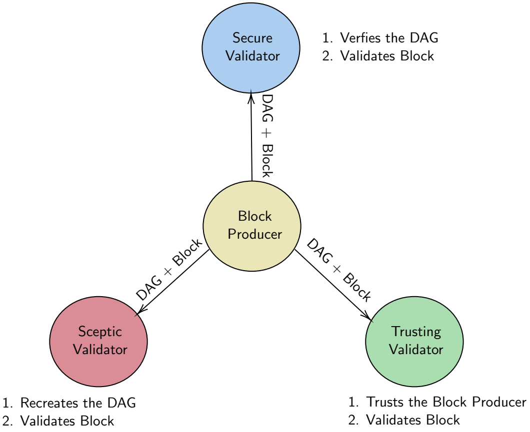


Figure: Possible validators in the blockchain

Related Work

Consensus protocols and transaction throughput are the two significant bottlenecks of blockchain performance.

Consensus:

- Proof of Stake *(PoS)*[9]
- Proof of Elapsed Time *(PoET)*[10]
- Practical Byzantine Fault Tolerance *(PBFT)*[11]

[9] **Vasim:2014:Blackcoin**.
[10] **sawtooth:url**.
[11] **Castro:2002:ACM**.

Consensus protocols and transaction throughput are the two significant bottlenecks of blockchain performance.

Sharding:

- Network Sharding[12]
- State Sharding[13]
- Transaction Sharding[14]

[12] **DiPETrans:CPE:2022**.
[13] **Zheng+:IEEE:TII:2022**.
[14] **Dang+:SIGMOD:2019**.

Table of Contents

Related Work

Consensus protocols and transaction throughput are the two significant bottlenecks of blockchain performance.

---

[15] **Dickerson+:ACSC:PODC:2017**.
[16] **Parwat+:Netys:2020**.
[17] **blockSTM:2022**.

Consensus protocols and transaction throughput are the two significant bottlenecks of blockchain performance. **Transaction throughput:**

STMs

- Block Graph (BG)[15]
- BG + Smart Validator[16]
- BlockSTM[17]

[15] **Dickerson+:ACSC:PODC:2017**.
[16] **Parwat+:Netys:2020**.
[17] **blockSTM:2022**.

DAG Implementations

- We have developed two implementations for the proposed DAG scheduler:
  - Linked List
  - Adjacency Matrix

Adjacency Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In-Degree Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 2 & 1 & 1 & 2 \end{bmatrix}$$

conflict parameters

In order to assess the framework's performance across different scenarios, we have devised three conflict parameters (CP):

- CP1 measures the proportion of transactions in the DAG that have at least one dependency
- CP2 represents the ratio of dependencies to the total number of transactions in the DAG
- CP3 quantifies the number of disjoint components, which are subgraphs without interconnections in the DAG.

Experiment Design

Parameter that we can change in Sawtooth architecture:

- Number of blocks
- Number of transactions in a block
- Metric for degree of dependency
- Number of threads