Efficient DAG-Based Consensus

FAB 22

Alberto Sonnino
Byzantine Fault Tolerance

> 2/3
How to build (really) high performance blockchains

The goal of this talk
Monolithic protocol sharing transaction data as part of the consensus

Optimize overall message complexity of the consensus protocol
Current Designs

Typical leader-based protocols
Current Designs
Typical leader-based protocols
Data dissemination is the key

Reaching consensus on metadata is cheap
Narwhal

Dag-based mempool
Narwhal
The workers and the primary

Client transactions

Narwhal mempool

Worker 1

Worker 2

Primary

Worker n
Narwhal
The workers and the primary
Narwhal
The workers and the primary

Client transactions

Narwhal mempool

Worker 1

Worker 2

Worker n

Transactions

Transactions

Transactions

Transactions

Batch

Batch

Batch

Primary
Narwhal
The workers and the primary
Narwhal
The workers and the primary

Client transactions

Narwhal mempool

Worker 1

Worker 2

Worker n

Transactions

Transactions

Transactions

Batch

Batch

Batch

Digest

Digest

Digest

'mempool protocol'
Narwhal
The primary machine
Narwhal
The primary machine
Narwhal
The primary machine
Narwhal
The primary machine

Round 1
Byzantine 'Reliable' Broadcast
Narwhal
The primary machine

r1  r2  r3  r4  r5

Diagram showing connections between nodes r1 to r5.
• The workers ship batch of transactions
• Many workers to scale out and use resources concurrently
• The primary constantly broadcasts the batch digests
• Headers at round $r$ contains references to $2f+1$ certificates of round $r-1$
• Build a structured DAG of certificates
Tusk

Zero-message asynchronous consensus
Tusk
Add common coin & Interpret the DAG
Tusk
The random coin elects the leader of r-2

r1  r2  r3

L1  coin
Tusk

The leader needs $f+1$ links from round $r-1$

Not enough support!
(Nothing is committed at this stage)
Tusk
Nothing is committed and we keep build the DAG
Tusk
Elect the leader of r3
Tusk

Leader L2 has enough support
Tusk
Leader L2 has links to leader L1

First commit L1
Then commit L2
Tusk
Commit all the sub-DAG of the leader
HotStuff on Steroids

Just by replacing the mempool
HotStuff on Narwhal
Overview

- Client transactions to Narwhal mempool
- Certificates to Partially Synchronous Consensus (HotStuff)
- Garbage collection
- Ordered transactions to State machine replication execution
HotStuff on Narwhal
Enhanced commit rule

C1
HotStuff on Narwhal
Enhanced commit rule
HotStuff on Narwhal
Enhanced commit rule
Evaluation

How to properly benchmark consensus protocols
Implementation

- Written in Rust
- Networking: Tokio (TCP)
- Storage: RocksDB
- Cryptography: ed25519-dalek

https://github.com/asonnino/narwhal
Evaluation
Typical mistakes

 herramienta persistent storage

ราว Do not sanitise messages

ราว Local/LAN benchmark + ping

ราว Many nodes on same machine

ราว Change parameters across runs

ราว Set transaction size to zero

ราว Preconfigure nodes with txs

ราว Send a single burst of transactions

ราว Benchmark for a few seconds

ราว Start timer in the batch maker

ราว Evaluate latency w/ only the first tx

ราว Separate latency and throughput

ราว Only benchmark happy path
Evaluation
Experimental setup on AWS
Evaluation

Typical mistakes

WillDisappear persistent storage

Do not sanitise messages

Local/LAN benchmark + ping

Many nodes on same machine

Change parameters across runs

Set transaction size to zero

Preconfigure nodes with txs

Send a single burst of transactions

Benchmark for a few seconds

Start timer in the batch maker

Evaluate latency w/ only the first tx

Separate latency and throughput

Only benchmark happy path
Evaluation
Set the benchmark parameters

Faults: 0 node(s)
Committee size: 10 node(s)
Transaction size: 512 B
Evaluation
Set the benchmark parameters

Faults: 0 node(s)
Committee size: 10 node(s)
Transaction size: 512 B

Header size: 1,000 B
Max header delay: 200 ms
GC depth: 50 round(s)
Sync retry delay: 5,000 ms
Sync retry nodes: 3 node(s)
batch size: 500,000 B
Max batch delay: 200 ms
Evaluation

Typical mistakes

 nhấn

Forgo persistent storage

Do not sanitise messages

Local/LAN benchmark + ping

Many nodes on same machine

Change parameters across runs

Set transaction size to zero

Preconfigure nodes with txs

Send a single burst of transactions

Benchmark for a few seconds

Start timer in the batch maker

Evaluate latency w/ only the first tx

Separate latency and throughput

Only benchmark happy path
Evaluation
Benchmark clients

Fixed input rate

For a long time (minutes)
Evaluation
Typical mistakes

้ำ Forgo persistent storage
้ำ Do not sanitise messages
้ำ Local/LAN benchmark + ping
้ำ Many nodes on same machine
้ำ Change parameters across runs
้ำ Set transaction size to zero
้ำ Preconfigure nodes with txs

้ำ Send a single burst of transactions
้ำ Benchmark for a few seconds
้ำ Start timer in the batch maker
้ำ Evaluate latency w/ only the first tx
้ำ Separate latency and throughput
้ำ Only benchmark happy path
Evaluation

Typical mistake

Load txs from pre-populated store & commit

Load txs from pre-populated store & commit

Propose batch 5 (pointer)
Evaluation
Typical mistakes

.createFromLeft

 Tümüne bakın! 🙄

- Forgo persistent storage
- Do not sanitise messages
- Local/LAN benchmark + ping
- Many nodes on same machine
- Change parameters across runs
- Set transaction size to zero
- Preconfigure nodes with txs

- Send a single burst of transactions
- Benchmark for a few seconds
- Start timer in the batch maker
- Evaluate latency w/ only the first tx
- Separate latency and throughput
- Only benchmark happy path
Evaluation
Typical mistake

send 50k txs (once)

Benchmark client → Narwhal mempool → Tusk

Ordered transactions

output after 400 ms

 достижени

TPS = \frac{50k}{400ms} = 125k \text{ tx/s}
Evaluation
Instrument the codebase

bench_start_time
sample_tx_id -> send_time
**Evaluation**

**Instrument the codebase**

batch_digest -> sample_tx_id

batch_digest -> batch_bytes

bench_start_time

sample_tx_id -> send_time
Evaluation
Instrument the codebase

batch_digest -> sample_tx_id
batch_digest -> batch_bytes
block_digest -> batch_digest

bench_start_time

sample_tx_id -> send_time

Narwhal mempool

Ordered transactions

Batch Maker
Proposer
Tusk
Evaluation

Instrument the codebase

batch_digest -> sample_tx_id
batch_digest -> batch_bytes
block_digest -> batch_digest

Benchmark client
Batch Maker
Proposer
Tusks

bench_start_time
sample_tx_id -> send_time

Ordered transactions

block_digest -> commit_time
**Evaluation**

*Compute throughput*

- `total_time = last_commit_time - bench_start_time`
- `BPS = total_bytes / total_time`
- `TPS = BPS / transaction_size`
**Evaluation**

**Compute latency**

\[
samples = \text{commit\_time} - \text{send\_time}
\]

\[
\text{latency} = \text{average}(samples)
\]
Evaluation
Typical mistakes

- Forgo persistent storage
- Do not sanitise messages
- Local/LAN benchmark + ping
- Many nodes on same machine
- Change parameters across runs
- Set transaction size to zero
- Preconfigure nodes with txs

- Send a single burst of transactions
- Benchmark for a few seconds
- Start timer in the batch maker
- Evaluate latency w/ only the first tx
- Separate latency and throughput
- Only benchmark happy path
Evaluation
Throughput latency graph
Evaluation

Throughput latency graph

Change only input rate
Evaluation
Throughput latency graph
Evaluation
Throughput latency graph
Evaluation
Throughput latency graph

Longer benchmarks
Evaluation
Throughput latency graph

Breaking point!
Evaluation
Typical mistakes

쫭 Forgo persistent storage
 vinden Do not sanitise messages
 vinden Local/LAN benchmark + ping
 vinden Many nodes on same machine
 vinden Change parameters across runs
 vinden Set transaction size to zero
 vinden Preconfigure nodes with txs

 vinden Send a single burst of transactions
 vinden Benchmark for a few seconds
 vinden Start timer in the batch maker
 vinden Evaluate latency w/ only the first tx
 vinden Separate latency and throughput
 vinden Only benchmark happy path
Evaluation

Throughput latency graph

[Graph showing throughput vs latency for different systems and configurations.]
Evaluation

Throughput latency graph
Evaluation

Scalability
Evaluation
Scalability
Evaluation

Scalability
Evaluation
Performance under faults
Evaluation

Typical mistakes

ıkl Forgo persistent storage
ıkl Do not sanitise messages
ıkl Local/LAN benchmark + ping
ıkl Many nodes on same machine
ıkl Change parameters across runs
ıkl Set transaction size to zero
ıkl Preconfigure nodes with txs

ıkl Send a single burst of transactions
ıkl Benchmark for a few seconds
ıkl Start timer in the batch maker
ıkl Evaluate latency w/ only the first tx
ıkl Separate latency and throughput

kl Only benchmark happy path
Evaluation

Still many caveats

- Perfect load balance
- Transaction deduplication
- Synthetic load
- No Byzantine adversary
- No network adversary
- Only AWS network
Conclusion

Narwhal & Tusk

• Separate consensus and data dissemination for high performance
• Scalable design, egalitarian resource utilisations

• Code: https://github.com/asonnino/narwhal
Acknowledgements

George Danezis
Lefteris Kokoris-Kogias
Alexander Spiegelman
Alberto Sonnino

Work done at Facebook Novi
Future Works
Come talk to us!

• Performance under DDoS attack?
• How to implement scalable execution?
Primary Implementation