A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform

João Sousa, Alysson Bessani, Marko Vukolić*

Faculdade de Ciências, Universidade de Lisboa

*IBM Research Zurich

Technical report available at
https://arxiv.org/abs/1709.06921
Outline

• Blockchain
• Hyperledger Fabric (HLF)
  – Protocol
• BFT-SMaRt Ordering Service
  – Architecture
  – Evaluation
• Conclusions
Blockchain

- A blockchain is an open database that maintains a distributed ledger typically deployed within a peer-to-peer network.
- Comprised by a continuously growing list of records called **blocks** that contain **transactions**.
- Blocks are protected from tampering by **cryptographic hashes** and a **consensus algorithm**.

\[
Hash_{\text{Block } i} = H(\text{Block } i-1)
\]
Blockchain

- Blockchains abide to either the *permissionless* or *permissioned* models
- **Permissionless**: A block can be added to the blockchain by any process, but only if a cryptographic puzzle is solved (Proof-of-Work)
- **Permissioned**: Closed group of processes execute a (traditional) *Byzantine consensus algorithm* to add a block to the blockchain
Blockchain

• Growth of blockchain technology made evident the performance limitations of the Proof-of-Work technique
  – 7 transactions per second and latencies of up to one hour (Bitcoin)

• Permissioned blockchain platforms try to avoid these limitations by using traditional Byzantine consensus algorithms
Hyperledger Fabric (HLF)

- Modular permissioned blockchain platform
- Enables clients to manage transactions by using *chaincodes*, *endorsing peers* and an *ordering service*
  - **Chaincode**: HLF’s counterpart for smart contracts
  - **Peers**: maintain the ledger/database and execute chaincodes and validate results
  - **Ordering service**: creates blocks of transactions and define the order by which they are added to the ledger
Hyperledger Fabric (HLF)

Fabric v1.0 Architecture

- **Application**
  - SDK
  - keys

- **Membership**
  - No SPoF
  - No SPoT

- **Peer**
  - Endorser
  - Committer
  - Ledger
  - Events
  - Chaincode

- **Orderer**
  - Order TXs in a batch according to consensus

- **Transactions**
  - 0 Enroll
  - 1 Proposal
  - 2 Submit Transaction
  - 3 Relay
  - 4 Batch
1. Create transaction and send it to endorsing peers
2. Transaction execution, endorsement, and sign
3. Collect endorsements and assemble proposal
4. Broadcast proposal
5. Verify endorsement
6. Client notified
Hyperledger Fabric (HLF)

- HLF supports different ordering services
- Current codebase (1.0) provides two:
  - Centralized module (Solo) requires very few hardware resources, but it is a single point of failure. Used mostly for testing.
  - Apache Kafka-based module (Kafka) is both decentralized and robust, but can only withstand crash faults
BFT-SMaRt

http://bft-smart.github.io/library/

- Byzantine Fault tolerant state machine replication library written in Java (developed at FCUL)
- Tolerates either crash ($2f+1$ replicas) or Byzantine faults ($3f+1$ replicas), under a partially sync. system model
- Being used in many academic projects and in some permissioned blockchains
BFT-SMaRt Ordering Service

• Uses the BFT-SMaRt replication library to provide Byzantine consensus
• Composed by an *ordering cluster* and a set of *frontends*
  – Ordering cluster comprised by $3f + 1$ nodes that create signed blocks with submitted transactions
  – Frontends received transactions from peers and submit them to the ordering cluster
BFT-SMaRt Ordering Service

Architecture

- Interface for the HLF codebase to submit transactions.
- Receives transactions from the consenter and relays them to the BFT-SMaRt proxy.
- Collects blocks from the BFT-SMaRt proxy.
- Signs blocks and submits them to the replica’s communication system.
- Submits/fetches transactions to/from the blockcutter and assembles the next block.
- Relays transactions to the ordering cluster and receives ordered blocks.
- Receives a stream of totally ordered transactions.

HLF Consenter
HLF SDK
Blockcutter
Signing & Sending Thread
Replica

Thread
Recv Thread
Threads

BFT-SMaRt Ordering Service

- Frontends collect $2f+1$ matching blocks signed from different ordering nodes
  - Guarantees a minimum of $f+1$ valid signatures
- HLF Java SDK is used to parse and assemble data structures and generate cryptographic hashes/signatures used by the platform
- Blocks can be signed in parallel without incurring in non-determinism
  - Blocks are assembled sequentially in the node thread
Evaluation

• Factors at play:

- Workload by Clients
- Tx size ($es$)
- Ordering Cluster size ($n$)
- Block size ($bs$)
- Number of Receiving Frontends ($r$)
Evaluation

• Throughput ($TP_{os}$) is bounded by either:
  – BFT-SMaRt ordering throughput ($TP_{bftsmart}$)
  – Number of blocks signed per second ($TP_{sign}$)

\[
TP_{os}^{bs,es,r} \leq \min(TP_{sign} \times bs, TP_{bftsmart}^{bs,es,r})
\]

*bs*: block size
*es*: transaction size
*r*: number of frontends
Evaluation

(Signature parallelization)

- Dell PowerEdge R410 machine (two quad-core 2.27 GHz Intel with hyper-threading) with 32 GB of memory
- Up to 16 hw threads with blocks containing 10 transactions of 0 bytes each

Upper bound of 84,000 transactions/seconds!
Evaluation
(Ordering Cluster - LAN)

• Dell PowerEdge R410 servers connected through a Gigabit ethernet switch
• Multiple frontends emulated with BFT-SMaRt clients
• Ordering Clusters of 4, 7, and 10 nodes
• Blocks of 100 transactions of different sizes: 40, 200, 1k, and 4k bytes
Evaluation
(Ordering Cluster - LAN, 4 nodes)
Evaluation
(Ordering Cluster - LAN, 7 nodes)
Evaluation

(Ordering Cluster - LAN, 10 nodes)
Evaluation
(Ordering Cluster - LAN, main findings)

• Impact of the number of frontends is negligible for larger transactions (1k and 4k bytes)
  – For smaller transaction (almost impossible to have in current HLF architecture), the transmission of blocks becomes the predominant bottleneck

• Even with blocks of 400k bytes, 32 frontends and a cluster of 10 nodes, the service reaches a peak throughput of ~2200 txs/sec
  – Twice more than Ethereum’s theoretical peak of 1000 txs/sec, and vastly superior to Bitcoin’s 7 txs/sec
Evaluation

(Ordering Cluster - geo-distributed)

• Focus on latency measurements in Amazon EC2 (VMs of type \textit{m4.4xlarge})
• 4 frontends (Canada, Oregon, Virgina, São Paulo)
• Moderate workload of at least 1000 txs/sec
• Replicas in Oregon, Ireland, Sydney, São Paulo, and Virginia
• Evaluated with standard BFT-SMaRt and WHEAT (a BFT-SMaRt optimized for wide-area)
Evaluation
(Ordering Cluster - geo-distributed)

(a) Canada (clients only).
(b) Oregon (weighted $V_{max}$, leader node).
(c) Virginia (weighted $V_{max}$).
(d) São Paulo (weighted $V_{min}$).
Evaluation
(Ordering Cluster - geo-distributed)

• WHEAT’s latency is consistently lower than standard BFT-SMaRt’s across all frontends by almost 50%

• Transaction size has a relatively minor impact on latency
  – Difference between a 40 and a 4k bytes transactions was never above 29 milliseconds (i.e., +10%)

• Placement of frontends can exhibit a larger impact on latency
  – As high as a 100 milliseconds difference using WHEAT (São Paulo vs Oregon)
Conclusions

• Ordering service module for the Hyperledger Fabric blockchain platform powered by BFT-SMaRt

• Practical evaluation performed in a local cluster and a geo-distributed environment
  – Lowest results for throughput still significantly superior to other platforms such as Ethereum and Bitcoin
  – Latency in wide-area displays values within half a second under moderate workload using WHEAT