Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

Blockchain Security Seminar
Pirmin Schmid
Bitcoin-like blockchains

- Distributed public anonymous ledger
- Consensus by longest chain
- PoW / PoS
- Fixed system for each variant

- Applications
Fabric

- Open-source Framework to build blockchains
- Modular for all aspects of the system
- Permissioned
- No currency
- Go, Java, Node.js, ...

- Example use cases

- New very crucial insights
Fabric Components

Membership service provider (MSP)

Endorser: execute
Committer: validate
Ledger: transaction manager (PTM)
KVS: Database

Docker

Policies
Chaincode
Fabric Components

Membership service provider (MSP)

Peer

- **Endorser**: execute
- **Committer**: validate
- **Ledger**: transaction manager (PTM)
- **KVS**: Database

Order service

Client

Policies

Chaincode

Gossip

Client

Client

Client

Client

Client
Fabric Components

Membership service provider (MSP)

- Client
- Endorser: execute
- Committer: validate
- Ledger: transaction manager (PTM)
- KVS: Database

Order service

- Client
- Endorser: execute
- Committer: validate
- Ledger: transaction manager (PTM)
- KVS: Database

Policies

- Client
- Chaincode

Gossip

- Client
Fabric Building blocks

- Store: CouchDB / LevelDB
- Chaincode: Go, Java, Node.js, …
- Docker containers
- gRPC
- Gossip: push/pull methods

- Orderer
  - Apache Kafka (ZooKeeper)
  - Byzantine Fault Tolerant (BFT) orderer
  - Solo (centralized) for development
Traditional Architecture

- Order by longest chain or BFT
- Execute smart contracts on all peers
- State updates on all peers → Ledger
Problem

- Sequential execution of all contracts on all peers → bottleneck
Problems

- Sequential execution of all contracts on all peers → bottleneck
- Programs MUST be deterministic → NO general purpose languages
Deterministic?

```
package main

import (
    "fmt"
)

func main() {
    m := []int{1, 2, 3, 4}
    for _, v := range m {
        fmt.Println(a: "Value:", v)
    }
}
```
Deterministic?

```go
code
package main

import (
    "fmt"
)

func main() {
    m := []int{1, 2, 3, 4}
    for _, v := range m {
        fmt.Println(a: "Value:", v)
    }
}
```

Value: 1
Value: 2
Value: 3
Value: 4
Deterministic?

```go
package main

import (   "fmt"
)

func main() {   m := map[int]string{1: "one", 2: "two", 3: "three", 4: "four"}
   for k, v := range m {     fmt.Println( a: "Key:", k, "Value:", v)
   }
}
```
Deterministic?

```go
package main

import (
    "fmt"
)

func main() {
    m := map[int]string{1: "one", 2: "two", 3: "three", 4: "four"}
    for k, v := range m {
        fmt.Println(a: "Key:", k, "Value:", v)
    }
}
```
Problems

- Sequential execution of all contracts on all peers → bottleneck
- Programs MUST be deterministic → NO general purpose languages
Fabric Architecture

Key insight
State

- Versioned key-value store
- Maintained on all peers
Execute

- Only some peers are executing the chaincode (simulation)
- Use current local state
- Create read-set and write-set for access of versioned key-value store
- Create signed “endorsement”
Fabric Architecture

Key insight

State must be replicated on all peers, not execution

Sequential execution in $O(n)$ instead of $O(N)$

$n \ll N$

$N = $ computing steps

$n = $ size of read and write sets

Execute

- Only some peers are executing the chaincode (simulation)
- Use current local state
- Create read-set and write-set for access of versioned key-value store
- Create signed “endorsement”
Fabric Architecture

Order
- Needs enough endorsements with identical read-/write-sets
- Uses Apache Kafka, BFT or other methods
- Peer gossip
Validate

- Parallel
- All peers validate correctness of transaction based on policy
- NO execution of the chaincode
Fabric Architecture

Update state

- sequential
- Peer transaction manager (PTM)
- Checks again versions of the keys in readset mismatch → invalidate transaction
Transaction flow
Policy

- Number of endorsements
- Which endorser shall be used
- Execution limitations
- Validation rules

- Parallel chaincode execution
- Confidential chaincode
Security

- TLS for communication
- Classic membership service
- Signatures
- Docker for sandboxing

- Complex system
- Dependency on many 3rd party codes
Evaluation

- Fabcoin: UTXO
- VMs in one data center
- 2.0 GHz 16 vCPU VMs running Ubuntu with 8 GiB RAM and SSDs
- 1Gbps networking connections
- Orderer: Kafka with 3 ZooKeeper nodes, 4 Kafka brokers, 3 Fabric orderers
- 5 peers, all Fabcoin endorsers
- TLS for all connections
- Signatures with 256-bit ECDA scheme
- Node clocks synchronized by NTP
- MINT phase / SPEND phase
Block size
Scales with number of vCPUs
## Latency in detail

<table>
<thead>
<tr>
<th>Step Description</th>
<th>avg</th>
<th>st.dev</th>
<th>99%</th>
<th>99.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) endorsement</td>
<td>5.6 / 7.5</td>
<td>2.4 / 4.2</td>
<td>15 / 21</td>
<td>19 / 26</td>
</tr>
<tr>
<td>(2) ordering</td>
<td>248 / 365</td>
<td>60.0 / 92.0</td>
<td>484 / 624</td>
<td>523 / 636</td>
</tr>
<tr>
<td>(3) VSCC val.</td>
<td>31.0 / 35.3</td>
<td>10.2 / 9.0</td>
<td>72.7 / 57.0</td>
<td>113 / 108.4</td>
</tr>
<tr>
<td>(4) R/W check</td>
<td>34.8 / 61.5</td>
<td>3.9 / 9.3</td>
<td>47.0 / 88.5</td>
<td>59.0 / 93.3</td>
</tr>
<tr>
<td>(5) ledger</td>
<td>50.6 / 72.2</td>
<td>6.2 / 8.8</td>
<td>70.1 / 97.5</td>
<td>72.5 / 105</td>
</tr>
<tr>
<td>(6) validation (3+4+5)</td>
<td>116 / 169</td>
<td>12.8 / 17.8</td>
<td>156 / 216</td>
<td>199 / 230</td>
</tr>
<tr>
<td>(7) end-to-end (1+2+6)</td>
<td>371 / 542</td>
<td>63 / 94</td>
<td>612 / 805</td>
<td>646 / 813</td>
</tr>
</tbody>
</table>
# Latency in detail

<table>
<thead>
<tr>
<th></th>
<th>avg</th>
<th>st.dev</th>
<th>99%</th>
<th>99.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>endorsement</td>
<td>5.6 / 7.5</td>
<td>2.4 / 4.2</td>
<td>15 / 21</td>
</tr>
<tr>
<td>2</td>
<td>ordering</td>
<td>248 / 365</td>
<td>60.0 / 92.0</td>
<td>484 / 624</td>
</tr>
<tr>
<td>3</td>
<td>VSCC val.</td>
<td>31.0 / 35.3</td>
<td>10.2 / 9.0</td>
<td>72.7 / 57.0</td>
</tr>
<tr>
<td>4</td>
<td>R/W check</td>
<td>34.8 / 61.5</td>
<td>3.9 / 9.3</td>
<td>47.0 / 88.5</td>
</tr>
<tr>
<td>5</td>
<td>ledger</td>
<td>50.6 / 72.2</td>
<td>6.2 / 8.8</td>
<td>70.1 / 97.5</td>
</tr>
<tr>
<td>6</td>
<td>validation (3+4+5)</td>
<td>116 / 169</td>
<td>12.8 / 17.8</td>
<td>156 / 216</td>
</tr>
<tr>
<td>7</td>
<td>end-to-end (1+2+6)</td>
<td>371 / 542</td>
<td>63 / 94</td>
<td>612 / 805</td>
</tr>
</tbody>
</table>
Latency in detail

<table>
<thead>
<tr>
<th></th>
<th>avg</th>
<th>st.dev</th>
<th>99%</th>
<th>99.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) endorsement</td>
<td>5.6 / 7.5</td>
<td>2.4 / 4.2</td>
<td>15 / 21</td>
<td>19 / 26</td>
</tr>
<tr>
<td>(2) ordering</td>
<td>248 / 365</td>
<td>60.0 / 92.0</td>
<td>484 / 624</td>
<td>523 / 636</td>
</tr>
<tr>
<td>(3) VSCC val.</td>
<td>31.0 / 35.3</td>
<td>10.2 / 9.0</td>
<td>72.7 / 57.0</td>
<td>113 / 108.4</td>
</tr>
<tr>
<td>(4) R/W check</td>
<td>34.8 / 61.5</td>
<td>3.9 / 9.3</td>
<td>47.0 / 88.5</td>
<td>59.0 / 93.3</td>
</tr>
<tr>
<td>(5) ledger</td>
<td>50.6 / 72.2</td>
<td>6.2 / 8.8</td>
<td>70.1 / 97.5</td>
<td>72.5 / 105</td>
</tr>
<tr>
<td>(6) validation (3+4+5)</td>
<td>116 / 169</td>
<td>12.8 / 17.8</td>
<td>156 / 216</td>
<td>199 / 230</td>
</tr>
<tr>
<td>(7) end-to-end (1+2+6)</td>
<td>371 / 542</td>
<td>63 / 94</td>
<td>612 / 805</td>
<td>646 / 813</td>
</tr>
</tbody>
</table>
## Latency in detail

<table>
<thead>
<tr>
<th>Step</th>
<th>avg</th>
<th>st.dev</th>
<th>99%</th>
<th>99.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) endorsement</td>
<td>5.6 / 7.5</td>
<td>2.4 / 4.2</td>
<td>15 / 21</td>
<td>19 / 26</td>
</tr>
<tr>
<td>(2) ordering</td>
<td>248 / 365</td>
<td>60.0 / 92.0</td>
<td>484 / 624</td>
<td>523 / 636</td>
</tr>
<tr>
<td>(3) VSCC val.</td>
<td>31.0 / 35.3</td>
<td>10.2 / 9.0</td>
<td>72.7 / 57.0</td>
<td>113 / 108.4</td>
</tr>
<tr>
<td>(4) R/W check</td>
<td>34.8 / 61.5</td>
<td>3.9 / 9.3</td>
<td>47.0 / 88.5</td>
<td>59.0 / 93.3</td>
</tr>
<tr>
<td>(5) ledger</td>
<td>50.6 / 72.2</td>
<td>6.2 / 8.8</td>
<td>70.1 / 97.5</td>
<td>72.5 / 105</td>
</tr>
<tr>
<td>(6) validation (3+4+5)</td>
<td>116 / 169</td>
<td>12.8 / 17.8</td>
<td>156 / 216</td>
<td>199 / 230</td>
</tr>
<tr>
<td>(7) end-to-end (1+2+6)</td>
<td>371 / 542</td>
<td>63 / 94</td>
<td>612 / 805</td>
<td>646 / 813</td>
</tr>
</tbody>
</table>
Conclusion

Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

Elli Androulaki  Artem Barger  Vitaly Vortnikov  Christian Cachin  Konstantinos Christidis  Angelo De Caro  David Enyeart  Christopher Ferris  Gennady Laventman  Yacov Manevich  Srinivasan Muralidharan  Chet Murthy  Binh Nguyen  Manish Sethi  Gari Singh  Keith Smith  Alessandro Sorniotti  Chrysoula Stathakopoulou  Marko Vukolić  Sharon Weed Cocco  Jason Yellick

IBM

<EURO/SYS'18>

April 23-26, 2018, Porto, Portugal
Reserve slides for questions
Blockchain use cases

- Food-safety network
- Global shipping trade
- Enterprise asset management
- Foreign exchange netting
- Global cross-currency payments

- One size does not fit all
Modules: allow step-wise improvements

- **Docker**: container but not actually sandbox
  Google just presented gVisor these days → improved security

- **Orderer**: Currently weak part of the system
  → improved distributed BFT based order is being built

- **Execution / Validation**: Can be extended to various policies and advancements in research

- **Storage**: Improved DBs / KVS if available
Google gVisor: available for docker
Apache Kafka: a distributed streaming platform
Number of peers

- mint-throughput-peer-LAN
- spend-throughput-peer-LAN
- mint-throughput-peer-2DC
- spend-throughput-peer-2DC

The graph shows the non-endorsing peer throughput as a function of the number of peers.
## Distance between data centers

<table>
<thead>
<tr>
<th>Netperf to TK [Mbps]</th>
<th>HK</th>
<th>ML</th>
<th>SD</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>peak MINT / SPEND throughput [tps] (without gossip)</td>
<td>240</td>
<td>98</td>
<td>108</td>
<td>54</td>
</tr>
<tr>
<td>1914 / 2048</td>
<td>1914 / 2048</td>
<td>1914 / 2048</td>
<td>1389 / 1838</td>
<td></td>
</tr>
<tr>
<td>peak MINT / SPEND throughput [tps] (with gossip)</td>
<td>2553 / 2762</td>
<td>2558 / 2763</td>
<td>2271 / 2409</td>
<td>1484 / 2013</td>
</tr>
</tbody>
</table>

- 100 peers across 5 data centers