Experiences with BFT-SMaRt as a consensus substrate of Permissioned Ledgers

Alysson Bessani
This session:

EXPERIENCES WITH DEPLOYED BLOCKCHAINS
These companies claim blockchain could help fight climate change

Is it a breakthrough, or just a buzzword?

By Jeremy Deaton  Nexus Media  January 18, 2018

Blockchain’s Fight Against Fake News

By Steven Buchko

The Tech That’s Changing How Cities Help the Homeless

From mapping apps to the blockchain, new tools are intended to give cities the information they need to address this growing challenge.
Treating Cancer with Blockchain Computing

Computer processing has become an important tool for diagnosing and treating cancer. The idea is that with personalized treatments coming into a clinical setting, rapid analysis of a patient's data becomes more crucial. The same applies to research in finding more cures for cancer.

OncoPower uses blockchain, cryptocurrency to help users manage cancer

By Laura Lovett | March 19, 2018

Correction: An earlier version of this story included an incorrect funding amount received by OncoPower.

A new platform exclusively for cancer patients is now registering new users. Witty Healthy's newly launched platform, called OncoPower, uses blockchain technology to help patients keep track of their medical data across providers and offer users incentives.
Outline

• BFT-SMaRt
  – Overview
  – Performance

• BFT-SMaRt in Permissioned Ledgers
  – Symbiont
  – Hyperledger Fabric
  – R3 Corda

• Beyond BFT-SMaRt
Part 1

BFT-SMART
State Machine Replication

Safety: all replicas execute the same sequence of commands
Liveness: commands issued by correct clients are answered
BFT-SMaRt

http://bft-smart.github.io/library/ [Bessani et al. DSN’14]

- Byzantine Fault tolerant state machine replication library written in Java (under development since 2010)
- Tolerates either crash \((2f+1)\) replicas or Byzantine faults \((3f+1)\) replicas), under a partially synchronous system model
- Available under Apache license
BFT-SMaRt Ordering Protocols

VP-Consensus (Byzantine)

Request
Propose
Write
Accept
Reply

VP-Consensus (Crash)

Request
Propose
Accept
Reply
Durability in BFT-SMaRt
[Bessani et al. USENIX ATC’13]

• Techniques for efficient durability
  – Parallel Logging
  – Sequential checkpoints
  – Collaborative state transfer
BFT-SMaRt Reconfiguration

Initiated by the **View Manager** - a trusted client used by system administrators that adds/removes replicas

Inform P4 that it joined the group
BFT-SMaRt Performance
(gigabit Ethernet, no disks)

Figure 4.6: Peak sustained throughput of BFT-SMaRt (gigabit Ethernet, no disks) for CFT ($2f+1$ replicas) and BFT ($3f+1$ replicas) considering different workloads and group sizes.

Finally, it is also interesting to see that, with relatively big requests (1024 bytes), the difference between BFT and CFT tends to be very small, independently on the number of tolerated faults. Moreover, the performance drops between tolerating 1 to 3 faults is also much smaller with large payloads (both requests and replies).

Mixed workloads. Figure 4.7 reports the results of our experiment considering a mix of read and write requests. In the context of this experiment, the difference between reads and writes is that the former issues small requests (almost-zero size) but gets replies with payload, whereas the latter issues requests with payload but gets replies with almost zero size. This experiment was also conducted under a saturated system running 1600 clients.

We performed the experiment both for the BFT and CFT setups of BFT-SMaRt, using requests and replies with payloads of 100 and 1024 bytes. Similarly to the previous experiments, the CFT protocol outperforms its BFT counterpart regardless of the ratio of read to write requests by around 5 to 15%.

However, the observed behavior of the system regarding
Performance under “sporadic” events
Part 1

BFT-SMART IN PERMISSIONED LEDGERS
• Startup from NY with 40+ people
• Technology
  – Smart contracts on top of state machine replication
  – BFT-SMaRt ported to Go
• Our involvement
  – Never saw the code
  – We talked a lot about collaboration, but just helped them understand the code and debug the synchronization phase of the protocol
• Open-source blockchain project targeting (at least initially) the financial market

• Key idea: **there is no shared global ledger**
  – Instead, **there are many distributed ledgers**

“Facts” shared by Alice and Bob

“Fact” shared by Ed, Carl and Demi
• Only involved participants have to **execute** and **validate** the transaction

• A transaction is **committed** only if it achieve
  – **Validity consensus**: all involved participants need to validate and sign the transaction
  – **Uniqueness consensus**: requires a notary service
• Notary implements an insert-only key-value store that register all state “consumptions”
• Some specific transaction validation might be executed
• Multiple notaries might be used
• Open-source, modular, permissioned [EuroSys’18]
• Architecture: not all “peers” are equal
1. Create transaction, send it to *endorsing peers*

2. Transaction simulation, create signed endorsement with *writeset* and *readset*

3. Collect endorsements for the same writeset and readset

4. Broadcast endorsed transaction

5. Transaction validation and committing

**Total Order + Block Creation**

**Ordering service cluster**

**Peers**

**Client**

*endorsing peers*
• Fabric supports different ordering services modules for different types of consensus
• Current release (v1.2.0) provides two:
  – Centralized module (*Solo*)
  – Apache Kafka-based module (*Kafka*)
• No module for Byzantine consensus
BFT-SMaRt Ordering Service
[Sousa et al, DSN’18]

Ordering Cluster
• Uses the BFT-SMaRt replication library to provide Byzantine fault-tolerant total order
• Compatible with release v1.1 of Fabric
• Comprised of an ordering cluster and a set of frontends

Frontend 0
Frontend f
Frontend f+1

Ordering Node 0
Ordering Node 1
Ordering Node 3f₀ + 1

BFT-SMaRt ordering protocol

Go ↔ Java

Client 0 ↔ Frontend 0

Ordering Cluster

Client M ↔ Frontend f+1
BFT-SMaRt Ordering Service

Frontend

Fabric codebase (Go)

Java SDK

Recv Thread

BFT-SMaRt Proxy

Client Threads

Ordering Nodes

Java SDK

Blockcutter

Node Thread

Block Creation Threads

BFT-SMaRt Replica

Threads

Threads

Threads
BFT-SMaRt Ordering Service

• Node state (to be persisted and transferred):
  – the ordered transactions still in the blockcutter,
  – header of the last generated block, and
  – latest configuration block

• Blocks can be validated and signed in parallel without incurring in non-determinism

• Frontends collect $2f+1$ matching blocks signed from different ordering nodes
Evaluation

• Factors at play:

Workload by Clients → Tx size ($es$) → Ordering cluster size ($n$) → Block size ($bs$) → Number of Receiving Frontends ($r$)
LAN Evaluation

4 nodes ($f=1$)  

10 nodes ($f=3$)
Some takeaways

• (LAN) Even with blocks of 100 4kB-txs, 32 frontends and a cluster of 10 nodes, the service orders ~2200 txs/sec
  – This is considered a big network for Fabric
  – (illustrative) 2x more than Ethereum’s theoretical peak of 100 txs/sec, and vastly superior to Bitcoin’s 7 txs/sec

• (WAN) 5 sites in 4 continents can order 1kB-txs in < 400ms (w/ a load of 1000 txs/s)
Part 1

BEYOND BFT-SMART
Our Research Agenda

• **Robust BFT replication library**
  – Maintain a good basic implementation

• **Geo-replication**
  – Key BFT application: distributed trust

• **Scalability & Elasticity**
  – Increase performance dynamically w/ additional replicas

• **Diversity and Fault Independence**
  – How to withstand $f$ malicious faults

• **Design a simple blockchain “platform”**
  – How to go from BFT SMR to a Blockchain
Geo-Replicated State Machines
[Sousa & Bessani. SRDS’15]

• Key techniques:
  – More replicas
  – Weighted replication
  – Tentative execution
Elastic State Machine Replication

[Nogueira et al. IEEE TPDS’17]

- **Split state** to create more storage and processing capacity
- **Rebalance load** by migrating heavily accessed partition to another replica group
- **Merge state** to better use resources
Elastic State Machine Replication

The creation of a 4GB partition takes a bit more than 2 minutes

Hotspot starts! Workload increases 10x
Diversity Management

Calculates the risk of having a common weakness two replicas

Exchange one or more replicas to decrease the risk of common weaknesses.

Fetch vuln. & exploit info from NVD, ExploitDB, etc.
Diversity Management

![Graph showing throughput for different OS configurations]

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Cores</th>
<th>JVM</th>
<th>Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB14</td>
<td>Ubuntu 14.04</td>
<td>4</td>
<td>Java Oracle 1.8.0.144</td>
<td>15GB</td>
</tr>
<tr>
<td>UB16</td>
<td>Ubuntu 16.04</td>
<td>4</td>
<td>Java Oracle 1.8.0.144</td>
<td>15GB</td>
</tr>
<tr>
<td>UB17</td>
<td>Ubuntu 17.04</td>
<td>4</td>
<td>Java Oracle 1.8.0.144</td>
<td>15GB</td>
</tr>
<tr>
<td>OS42</td>
<td>OpenSuse 42.1</td>
<td>4</td>
<td>Openjdk 1.8.0.141</td>
<td>15GB</td>
</tr>
<tr>
<td>FE24</td>
<td>Fedora 24</td>
<td>4</td>
<td>Openjdk 1.8.0.141</td>
<td>15GB</td>
</tr>
<tr>
<td>FE25</td>
<td>Fedora 25</td>
<td>4</td>
<td>Openjdk 1.8.0.141</td>
<td>15GB</td>
</tr>
<tr>
<td>FE26</td>
<td>Fedora 26</td>
<td>4</td>
<td>Openjdk 1.8.0.141</td>
<td>15GB</td>
</tr>
<tr>
<td>DE7</td>
<td>Debian 7</td>
<td>4</td>
<td>Java Oracle 1.8.0.151</td>
<td>15GB</td>
</tr>
<tr>
<td>DE8</td>
<td>Debian 8</td>
<td>4</td>
<td>Openjdk 1.8.0.151</td>
<td>15GB</td>
</tr>
<tr>
<td>W10</td>
<td>Windows 10</td>
<td>4</td>
<td>Java Oracle 1.8.0.151</td>
<td>15GB</td>
</tr>
<tr>
<td>WS12</td>
<td>Windows Server 2012</td>
<td>4</td>
<td>Java Oracle 1.8.0.151</td>
<td>15GB</td>
</tr>
<tr>
<td>FB10</td>
<td>FreeBSD 10</td>
<td>4</td>
<td>Openjdk 1.8.0.144</td>
<td>15GB</td>
</tr>
<tr>
<td>FB11</td>
<td>FreeBSD 11</td>
<td>4</td>
<td>Openjdk 1.8.0.144</td>
<td>15GB</td>
</tr>
<tr>
<td>SO10</td>
<td>Solaris 10</td>
<td>1</td>
<td>Java Oracle 1.8.0.141</td>
<td>15GB</td>
</tr>
<tr>
<td>SO11</td>
<td>Solaris 11</td>
<td>1</td>
<td>Java Oracle 1.8.0.05</td>
<td>15GB</td>
</tr>
<tr>
<td>OB60</td>
<td>OpenBSD 6.0</td>
<td>1</td>
<td>Openjdk 1.8.0.72</td>
<td>15GB</td>
</tr>
<tr>
<td>OB61</td>
<td>OpenBSD 6.1</td>
<td>1</td>
<td>Openjdk 1.8.0.121</td>
<td>15GB</td>
</tr>
</tbody>
</table>
BFT-SMaRt as a Blockchain

• What to change?
  – Durable Logging -> Blockchain
  – State machine service-> smart contract
  – BFT reconfiguration -> Churn/committee management
  – VP-consensus -> Scalable VP-consensus
Questions?

- To know more: