- Permissioned Ledgers -
Consensus is Only the Beginning

Alysson Bessani
Who am I?

- Started PhD on Byzantine Replication (2002)
- Multi-cloud Storage (2010)
  - DepSpace (2008)
  - Spinning (2009)
  - MinBFT (2013)
  - Mod-SMaRt (2012)
  - BFT-SMaRt (2014)
  - WHEAT (2015)
- Fabric BFT Orderer (2017)
- The DiSIEM project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 700692.


http://vawlt.io
What is a Blockchain?

General ledger on top of a peer-to-peer network
Blockchain Models

• **Public (Open) Ledgers**
  – Like the ones used in **Bitcoin** and **Ethereum**
  – Peers don’t need strong identities
  – PoW/PoS/... consensus

• **Permissioned (Private) Ledgers**
  – a.k.a. **Distributed Ledger Technology (DLT)**
  – Peers have strong/verifiable identities
  – Classical Byzantine consensus
• Open-source, modular, permissioned
• Architecture: not all “peers” are equal
1. Create transaction, send it to *endorsing peers*

2. Create signed endorsement with write and read sets

3. Collect endorsements

4. Broadcast endorsed transaction

5. Transaction commit and validation

**Consensus + Block Creation**

**Ordering service**

**Peers**

**Client**

7/10/18 Chain-in
• Ordering node state:
  – the ordered transactions not yet in a block,
  – header of the last generated block, and
  – latest configuration block
• 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**
• 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
State Machine Replication

Safety: all replicas execute the same sequence of transactions

Liveness: transactions issued by correct clients are answered
BFT-SMaRt


- 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
- Available under Apache license
BFT-SMaRt Performance
(gigabit Ethernet, no disks)

Figure 4.6: Peak sustained throughput of BFT-SMaRt 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 the ratio of read to write requests is quite interesting. As shown in Figure 4.7, the performance of CFT is significantly higher than that of BFT, especially when the ratio of read to write requests is low. This is because CFT is designed to handle read requests more efficiently than write requests, as it uses a special protocol for reading from disk. In contrast, BFT-SMaRt uses a more general protocol that is optimized for writing to disk, which can lead to performance degradation when reading from disk.

In summary, the results of our experiments demonstrate the performance benefits of using BFT-SMaRt in a gigabit Ethernet environment with no disks. The CFT protocol outperforms its BFT counterpart, especially when handling mixed workloads with large payloads. Further research is needed to understand the underlying reasons for these performance differences and to optimize the protocol for even better performance in future systems.
Consensus is not enough

• A consensus engine also needs:
  
  – **Durability**: transactions on stable storage
  – **Recovery**: recovered replicas need to be synched
  – **Reconfiguration**: replica group changes
Durability = Stable Logging

Throughput (4kB-writes/sec)

Memory: 4772
Async Disk: 4312
Sync Disk: 63
Sync SSD: 1017
PROBLEM:
- Need 2 replicas to order requests
- 1 stopped and 1 transmitting state
Reconfiguration = Complexity

Features X Complexity

- Best case
- High performance
- Leader change
- Crash-recovery
- Production-level system
- Decent PhD-level prototype
- Accepted paper
- Rejected paper

SMR Complexity (LoCs & Module dependencies)

7/10/18
Durability in BFT-SMaRt

- Techniques for efficient durability
  - Parallel Logging
  - Sequential checkpoints
  - Collaborative state transfer
For large writes, our disks absorb more data than (our) SSDs. Disks can reach pure memory throughput.

<table>
<thead>
<tr>
<th>Record length</th>
<th>Disk</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1MB</td>
<td>96.1 MB/s</td>
<td>128.3 MB/s</td>
</tr>
<tr>
<td>4MB</td>
<td>135.6 MB/s</td>
<td>130.7 MB/s</td>
</tr>
</tbody>
</table>
BFT-SMaRt Performance under “sporadic” events

New replica enters the group

Leader crashes

New leader takes over

Old leader recovers

Replica removed from the group

7/10/18
BFT-SMaRt Ordering

Fabric codebase (Go)

Java SDK

Recv Thread

BFT-SMaRt Proxy

Client Threads

Blockcutter

Node Thread

Block Creation Threads

Java SDK

BFT-SMaRt Replica

Ordering Nodes

Frontend
BFT-SMaRt Ordering Evaluation

4 nodes \((f=1)\)

10 nodes \((f=3)\)

Throughput (ktrans/sec)

Number of receivers

Number of receivers

BFT-SMaRt Ordering Evaluation

It can be observed that when using 10 envelopes/block instead of only 10. Moreover, this configuration makes the BFT-SMaRt Ordering Service throughput for different envelope, block and cluster sizes.

The throughput becomes similar to the results observed in [5]. This happens because even though the overhead associated with BFT-SMaRt Ordering Service with asynchronous clients.

Hence, the performance can take up to 60% of CPU usage when executing a void service with asynchronous clients.

Moreover, for up to 8 receivers. This happens because even though the transmission of blocks to the receivers exhausts. It can be observed that, across all cluster sizes, BFT-SMaRt Ordering Service is able to produce signatures.

As discussed previously, the predominant overhead becomes

as the number of receivers (16 and 32), throughput converges to similar values across all combinations of envelope/cluster/block sizes.

As discussed previously, the predominant overhead becomes

as the number of receivers (16 and 32), throughput converges to similar values across all combinations of envelope/cluster/block sizes.

As discussed previously, the predominant overhead becomes

as the number of receivers (16 and 32), throughput converges to similar values across all combinations of envelope/cluster/block sizes.

As discussed previously, the predominant overhead becomes

as the number of receivers (16 and 32), throughput converges to similar values across all combinations of envelope/cluster/block sizes.

As discussed previously, the predominant overhead becomes

as the number of receivers (16 and 32), throughput converges to similar values across all combinations of envelope/cluster/block sizes.

As discussed previously, the predominant overhead becomes
A R&D 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
Questions?

- Alysson Bessani
  - anbessani@fc.ul.pt
  - www.di.fc.ul.pt/~bessani

To know more: