HotStuff: Three-step Rule!

Dahlia Malkhi
VMware Research
HotStuff
Blockchain: Decentralizing Trust

Create digital assets without backing liability  X
Book-keeping without backing liability
Permissioned BFT Consensus

until now..

Clement et al., 2009: "A single faulty client or server is capable of rendering PBFT, Q/U, HQ, and Zyzzyva virtually unusable"

Amir et al., 2011: “faulty processors can significantly degrade the performance of some protocols”

Cachin and Vukolic 2017: “Tendermint ... suffers from a livelock bug”

VMware team 2017: "a safety violation in Zyzzyva and a liveness violation in FaB. ... relatively simple scenarios, involving only four replicas, and one or two view changes."

IBM FaB, 2017: "version 1.0 was launched early in 2017 without an implementation of a BFT ordering service"

JPMC 2018: "Note: The QuorumChain consensus algorithm is not yet supported by this release"
BFT Properties

Safety
  • against $f$ out of $(n=3f+1)$ bad players and against asynchrony
  • against $f$ out of $(n=2f+1)$ bad players
  • hybrid

Liveness
  • against asynchrony, via randomization or GST

Efficient
  • linear under most conditions
  • leader based
HotStuff
HotStuff: Three-Step BFT Consensus Rules
A simple framework for blockchain BFT protocols

*Chain Quality*: maintain linearity against frequent proposer replacement

*Safety*: Against $f$ out of $3f+1$ and against asynchrony

*Responsiveness*: Advance at the network speed after GST
Quorum certificate (QC): $n-f$ votes

unique and live
One-chain QC
Iterate thru views

As leader,

wait for n-f NEWVIEWs, broadcast proposal

wait for n-f votes, broadcast QC

As replica,

send NEWVIEW with locked QC

wait for leader pr until leader timeout
if no lock \(^{\neg}pr\), send vote, lock pr

If receive QC, decide
One-chain QC:

Correct leader reaches decision,
And \( f+1 \) correct are locked

Incorrect leader leaves locks
Is it safe to unlock \( v \)? only if \( n-f \) have \( \checkmark v \)
Not less? No!
If \( 2f \) say \( v \) and another \( 2f \) say \( v' \) which one is safe?
Is it live to wait? No!
HotStuff
HotStuff

X

n-f votes
Quorum Certificate (QC): 
\((2f+1)\) of \((3f+1)\) votes

\((f+1)\) good \(f\) slow \(f\) bad
HotStuff

Quorum Certificate (QC):
$(2f+1)$ of $(3f+1)$ votes

$(f+1)$ good  $f$ slow  $f$ bad
One-chain QC:

Guarantees uniqueness
If you have it, you can prove uniqueness
If you don’t, you cannot convince a proposal is safe
One-chain QC:

What about [DLS88]?

Only a leader decides per view
Unlock during (complex) view-change protocol
Two-chain QC:

Correct leader reaches decision,
And \( f+1 \) correct are locked holding a QC

Incorrect leader leaves locks
If \( f+1 \) don’t have it, safe to unlock

Is it live to wait for \( f+1 \)? Yes!
Two QC
Iterate through views

As leader,
Unlock-Phase
wait for n-f NEWVIEWs
broadcast *proposal* with $f+1$-proof
Phase-1
wait for $n-f$ votes, broadcast *QC*
Phase-2
wait for $n-f$ votes, broadcast *QC[QC]*

As replica,

send NEWVIEW with locked QC
wait for leader *pr* until leader timeout
if received *pr*, send ACK
wait for leader *QC* until leader timeout
if received *QC*, send ACK, lock *QC*
If receive *QC[QC]*, decide
Two-chain QC:

Guarantees uniqueness
If you have it, you know uniqueness
And you can prove safety
Two-chain QC:

All communication steps are linear* except unlock step

*with signature-combining, otherwise, quadratic
Two-chain QC:

Tendermint: All communication steps are linear
Must wait maximal network delay
one block per period

synchronous: always slow

asynchronous: sometimes very slow/expensive

8,000,000,000 msgs
HotStuff
Three-chain QC:

Correct leader reaches decision,
And $f+1$ correct are locked

Correct locked if $f+1$ hold a QC

Incorrect leader leaves locks
Then $f+1$ have QC, enough to unlock

Is it live to wait for $f+1$? Yes!
Three QC
Iterate through views

As leader,

Unlock-phase
wait for n-f NEWVIEWs
broadcast *proposal* with highest lock

Phase-1
wait for \( n-f \) votes, broadcast \( QC \)

Phase-2
wait for \( n-f \) votes, broadcast \( QC[QC] \)

Phase-3
wait for \( n-f \) votes, broadcast \( QC[QC[QC]] \)

As replica,

send NEWVIEW with locked QC

wait for leader \( pr \) until leader timeout
if received \( pr \), send vote

wait for leader \( QC \) until leader timeout
if received \( QC \), send vote

wait for leader \( QC[QC] \) until leader timeout
if received \( QC[QC] \), send vote, lock \( QC \)

If receive \( QC[QC[QC]] \), decide
Three-chain QC:

All communication steps are linear*
No exception!

*with signature-combining, otherwise, quadratic
BFT solutions resilient against $f$ of $n=3f+1$, and against asynchrony

<table>
<thead>
<tr>
<th>year</th>
<th>protocol</th>
<th>Chain-quality</th>
<th>Responsive</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1976</td>
<td>PLS</td>
<td>O(n^3)</td>
<td>no</td>
<td>(synchronous)</td>
</tr>
<tr>
<td></td>
<td>Byz. Generals</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1988</td>
<td>DLS</td>
<td>O(n^3)</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>PBFT</td>
<td>O(n^3)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Zyzzyva</td>
<td>O(n^3)</td>
<td>yes</td>
<td>fast track</td>
</tr>
<tr>
<td>2016</td>
<td>Tendermint, Casper</td>
<td>O(n^2)</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>2017</td>
<td>ALGORAND</td>
<td>O(nM)</td>
<td>no</td>
<td>sortition</td>
</tr>
<tr>
<td>2018</td>
<td>SBFT</td>
<td>O(n^2)</td>
<td>yes</td>
<td>fast track + echo</td>
</tr>
<tr>
<td>2018</td>
<td>Hot-Stuff</td>
<td>O(n)</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
HotStuff: Better and simpler!
Pipelining Three QC
Iterate through views

As leader,
Unlock-phase
wait for $n-f$ NEWVIEWs
broadcast *proposal* with highest lock

Phase-1
wait for $n-f$ votes, broadcast *QC*

Phase-2
wait for $n-f$ votes, broadcast *QC[QC]*

Phase-3
wait for $n-f$ votes, broadcast *QC[QC[QC]]*
HotStuff: Pipelined Three QC
Iterate through views

As leader,
Unlock-phase
wait for \( n-f \) NEWVIEWs
broadcast proposal with highest lock
Phase-1
wait for \( n-f \) votes,
send next leader NEXTVIEW with QC

As replica,
send locked QC
wait for leader \( pr \) until leader timeout
if received \( pr \), send vote
for highest \( QC[QC] \), lock QC
for highest \( QC[QC][QC] \), decide

As next leader
Wait for NEXTVIEW until leader timeout
HotStuff
one-chain

2f+1 votes on a unique block
two-chain

2f+1 votes on a unique block
2f+1 nodes know there is a unique block
three-chain

2f+1 votes on a unique block
2f+1 nodes know there is a unique block
every one of 2f+1 nodes can convince it is unique
DLS [Dwork Lynch Stockmeyer, 1988]

PBFT [Castro and Liskov, 1999]

Tendermint [Buchman 2016]

Casper [Buterin and Griffin, 2017]
HotStuff [Yin, Malkhi, Reiter, Gueta, Abraham 2018]

https://arxiv.org/abs/1803.05069
Open-source code coming soon