**HotStuff: BFT Consensus with Linearity and Responsiveness**

by Shuyi (Leader), Gabriel Rodgers (Blogger), Sami Aljabery (Scribe)

**Introduction**

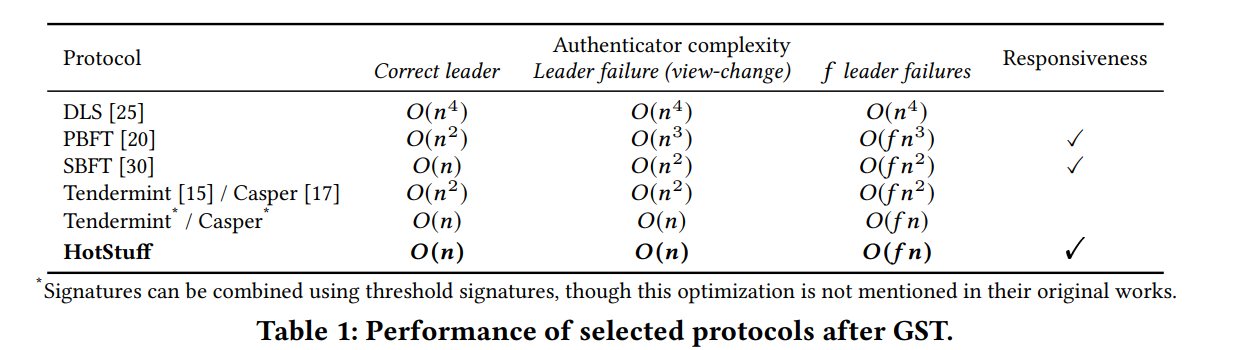
HotStuff is a Byzantine Fault Tolerance (BFT) consensus protocol that achieves much lower time complexity than previous BFT consensus implementations. HotStuff is implemented for partially synchronous systems. It achieves a lower time complexity by using a 4-step design, pipelining (in chained HotStuff), and message cryptography.

**Keywords**

* BFT: Byzantine Fault Tolerance; toleration of faulty/malicious nodes.
* Consensus: Multiple nodes coming to an agreement on a subject.

**Problem**

Current BFT protocols do not scale well for larger-node distributed systems (PBFT, SBFT, etc.); some protocols accomplish consensus in O(N^2) time, with N being the number of nodes. Hotstuff remedies this problem by having a time complexity of O(N) for consensus and O(N^2) for leader replacement.



**Implementation**

HotStuff uses Cryptography for messages: each node has a signature assigned to it, which is used to provide proof that a message comes from a given node. The signatures from each node are aggregated and merged into a Quorum Certificate (QC), which allows for a simple and fast way to see if a consensus has been reached or not. This decreases the number of signatures required to be authenticated to O(N) time.

HotStuff has 4 steps:

1. Prepare - HotStuff chooses the next view by having all nodes send their prepareQC values to the leader, and also different command proposals are shared.
2. Pre-Commit - HotStuff waits until n-f nodes agree on moving a proposal to the pre-commit stage, then combines agreements into a QC and sends it to all nodes.
3. Commit - HotStuff broadcasts precommit QC to all nodes, then nodes vote on moving to Commit stage. If n-f nodes agree to move to the Commit stage, then they are locked into that commit QC.
4. Decide - Hotstuff moves to the next view.

The chained implementation of HotStuff allows for pipelining - a set of nodes can be in different views at the same time, with each view being in a different HotStuff stage. This allows for more parallelism and better performance.

**Discussion**

Q: Is most of the model based on the basic hotstuff or the chained hotstuff?

A: I feel like the chained hotstuff is improved by the basic hotstuff. In the slides, we can see how chained hotstuff performs in comparison to basic hotstuff.

Q: Can a leader election happen between the two command stages? Does it ever happen?

A: I think we can have different leaders between commands. The table showcases a pipeline happening.

Q: On Slide 14, what does optimistic response mean?

A: It makes sure that even with a faulty leader, it can elect a new leader it can keep going.

Q: For all the protocols being tested in the table, are all the protocols in the table all partially synchronous models or partially asynchronous?

A: They can’t all be asynchronous, it’s more likely to be synchronous.

Q: Is this still being used in industry?

A: It was used for quite a bit at Facebook, it’s still recent.

Q: For a distributed system that requires consistent data, would BFT be the beneficial choice?

A: If your system and nodes are secured, you probably don’t need BFT.

**Sources**

<https://dl.acm.org/doi/10.1145/3293611.3331591>

<https://courses.corelab.ntua.gr/pluginfile.php/9663/course/section/1387/22-23.atc.balla.pdf>