Building a Performance Model of the *Tendermint* Concensus Algorithm

Jonas Vanden Branden

jonas.vandenbranden@student.uantwerpen.be

Abstract

In the current landscape of blockchain development, we are now confronted with many limit of blockchains and distributed applications. We investigate Tendermint, a blockchain concensus platform provides '*Byzantine fault-tolerant replicated state machines in any programming language*'. When working with new technology and algorithms, it is educational to perform thorough research on all aspects of these new systems to ensure correct and efficient use. In this paper, the first steps are taken for modeling the Proof-of-stake Tendermint Concensus algorithm in the light of performance. 'Stochastic Activity Networks' are found to be a very interesting formalism for modeling distributed concensus algorithms in related work.

Keywords: Blockchain, Performance, SAN, Stochastic Activity Networks, Modeling, Mobius, Tendermint, Concensus

1. Introduction

At present, the performance of the this concensus-algorithm is not really studied by modeling, although some experiments are performed by its creators, as seen in figure 1. This can be an interesting starting point to compare the early simulation results against real world experimental results.

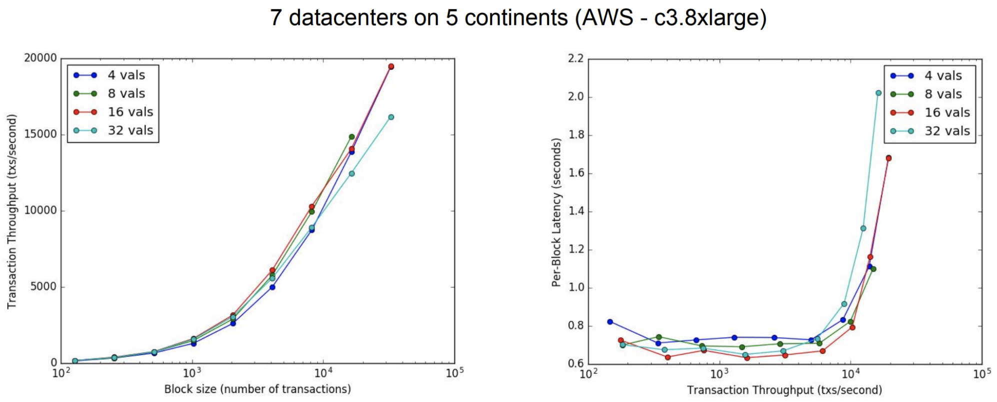

Figure 1: Experimental performance results in [3]

The *Byzantine Consensus Algorithm* itself is descibed in details in the github[1] documentation of Tendermint. In addition to the detailed explanation of how it works, there is a state machine described which is used for the nodes participating in the network. (see Fig. 4) The Tendermint platform provides an application blockchain interface (ABCI) for developers to 'plugin' their application logic into the Tendermint concensus engine.

In this paper, we will explore the possibilities of modeling such algorithm effectively. In the second chapter, we will investigate the reasons for modeling this system. In the third chapter, the formalism used is discussed in relation to alternatives and related work. In the fourth chapter, the software tool used (Möbius) is proposed. In the fifth chapter, the concensus algorithm is dissected and partly modeled as a Hiërarchical Stochastic Activity Network. In chapter six, the conclusions are drawn, the most important obstacles are named and ideas for future work are suggested.

2. Why Modelling?

Having a model of the concensus algorithm to simulate performance in various scenario's would be interesting for both the low level concensus developers (to have a good way of predicting impact of algorithm changes on performance) as the higher level application developers (to have a way of simulating general throughput/latency performance when scaling up the infrastructure running their application).

3. Formalism of choice: Stochastic Activity Networks

When approaching this problem of modeling the distributed model, a fundamental question is which formalism to use to represent the concensus algorithm. (Stochastic) Reward Nets are an interesting way of modeling such situations, as the authors of [10] have done when modeling the Hyperledger Fabric pBFT concensus process, as can be seen in Fig. 2, where a situation of three processes is modeled.

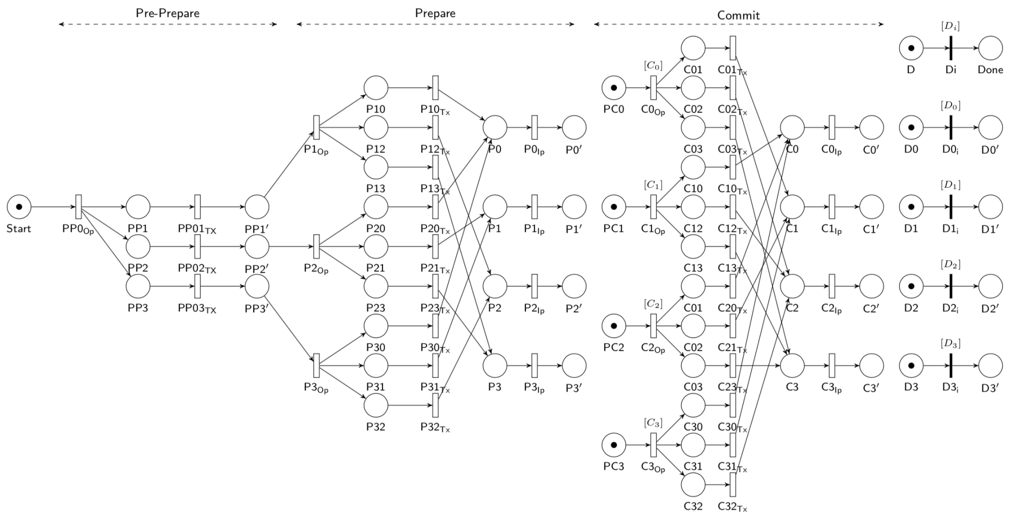

Figure 2: SRN model of Hyperledger Concensus

However, this kind of modeling gets complex quickly, due to lack of hierarchical structures and reusability, and limited expressiveness. As previous research has indicated ([5]), there is another formalism (also an extension on

SPN's) that is better at representing such processes; (Hierarchical) Stochastic Activity Networks (or SAN's). These networks extend SPN's with two elements; Input and Output gates. More formal definitions and concepts are described in [4].

4. Tool of choice: Möbius

The Möbius tool [12] is a modeling tool developed at the University of Illinois, aimed at validation of system performance (among other aspects). It supports Stochastic Activity Networks and is free for educational use.
The tool enables creation of Atomic SAN's and multiple models can be composed using the Repl/Join formalism, which enables us to create multiple 'instances' of models (replication) to run and combine the states of multiple models (join). A reward model can be used to define the *scoring* model of an atomic or composed model.
A study model can be used to investigate the behavior of systems for several different parameter values, as a set, a range or a more advanced 'Design of Experiments'. Models can be 'solved' by either simulation or a formal approach.An overview of all the components can be seen in figure 3.

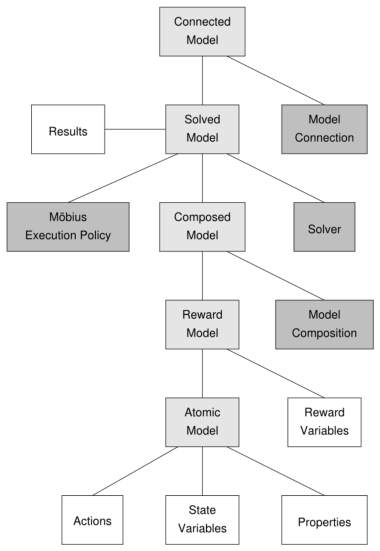

Figure 3: Möbius framework components [13]

5. Building the Model

We will dissect the Tendermint concensus algorithm analogous to the method used in chapter 6 of [7].

5.1. The Algorithm

As mentioned before, the algorithm is described extensively on the Tendermint github wiki [1]. We will follow the structure of that page to dissect the states of the state machine (see fig 4) that describes the algorithm.

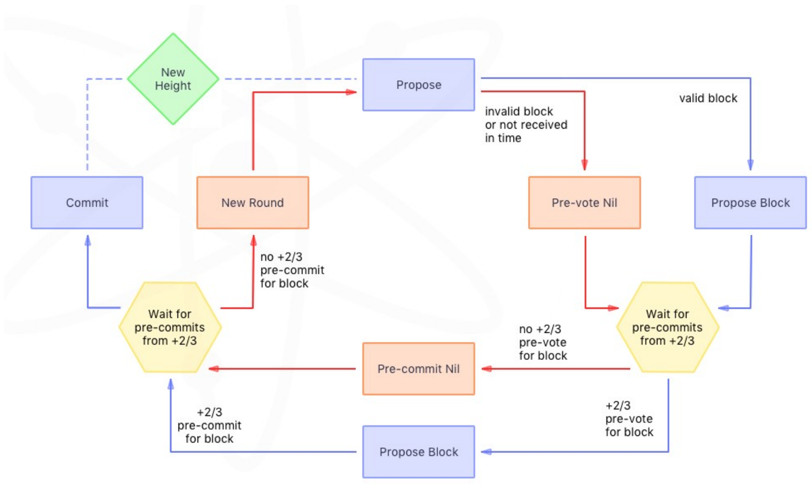
New Height
Propose
valid block
invalid block or not received in time
Commit
New Round
Pre-vote Nil
Propose Block
no +2/3 pre-commit for block
Wait for pre-commits from +2/3
no +2/3 pre-vote for block
Wait for pre-commits from +2/3
Pre-commit Nil
+2/3 pre-commit for block
+2/3 pre-vote for block
Propose Block

Figure 4: The algorithm state machine [2]

Terminology

- The network is composed of optionally connected **nodes**. Nodes directly connected to a particular node are called **peers**.
- The consensus process in deciding the next block (at some height H) is composed of one or many rounds. NewHeight, Propose, Prevote, Precommit, and Commit represent state machine states of a round. (aka RoundStep or just "step").
- A node is said to be at a given height, round, and step, or at (H,R,S), or at (H,R) in short to omit the step. To prevote or precommit something means to broadcast a prevote vote or precommit vote for something.

- A vote at (H,R) is a vote signed with the bytes for H and R included in its sign-bytes.
- +2/3 is short for ”more than 2/3”
- 1/3+ is short for ”1/3 or more”
- A set of +2/3 of prevotes for a particular block or $< nil >$ at (H,R) is called a proof-of-lock-change or **PoLC** for short.

Propose Step (Height H, Round R)

Upon entering Propose:

- The designated proposer proposes a block at (H,R).

The Propose step ends:

1. After timeoutProposeR after entering Propose.
   → goto Prevote(H,R)
2. After receiving proposal block and all prevotes at PoLC-Round.
   → goto Prevote(H,R)
3. After common exit conditions*

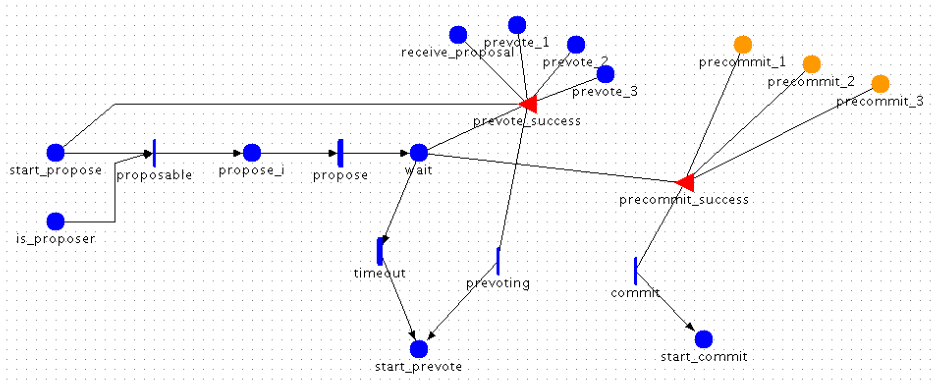

Figure 5: ProposeStep atomic SAN prototype

Prevote Step (Height H, Round R)

Upon entering Prevote, each validator broadcasts its prevote vote.

- First, if the validator is locked on a block since LastLockRound but now has a PoLC for something else at round PoLC-Round where $LastLockRound < PoLCRound < R$, then it unlocks.
- If the validator is still locked on a block, it prevotes that.
- Else, if the proposed block from Propose(H,R) is good, it prevotes that.
- Else, if the proposal is invalid or wasn't received on time, it prevotes $< nil >$.

The Prevote step ends:

1. After +2/3 prevotes for a particular block or $< nil >$.
   $\rightarrow$ goto Precommit(H,R)
2. After timeoutPrevote after receiving any +2/3 prevotes.
   $\rightarrow$ goto Precommit(H,R)
3. After common exit conditions*

Precommit Step (Height H, Round R)

Upon entering Precommit, each validator broadcasts its precommit vote.

- If the validator has a PoLC at (H,R) for a particular block B, it (re)locks (or changes lock to) and precommits B and sets LastLockRound = R.
- Else, if the validator has a PoLC at (H,R) for $< nil >$, it unlocks and precommits $< nil >$.
- Else, it keeps the lock unchanged and precommits $< nil >$.

A precommit for $< nil >$ means "I didn't see a PoLC for this round, but I did get +2/3 prevotes and waited a bit".

The Precommit step ends:

1. After +2/3 precommits for $< nil >$.
   $\rightarrow$ goto Propose(H,R+1)
2. After timeoutPrecommit after receiving any +2/3 precommits.
   $\rightarrow$ goto Propose(H,R+1)
3. After common exit conditions*

**Common exit conditions*

Common exit conditions and their actions are:

1. After +2/3 precommits for a particular block.
   → goto Commit(H)
2. After any +2/3 prevotes received at (H,R+x).
   → goto Prevote(H,R+x)
3. After any +2/3 precommits received at (H,R+x).
   → goto Precommit(H,R+x)

Commit Step (Height H)

- Set CommitTime = now()
- Wait until block is received
  → goto NewHeight(H+1)

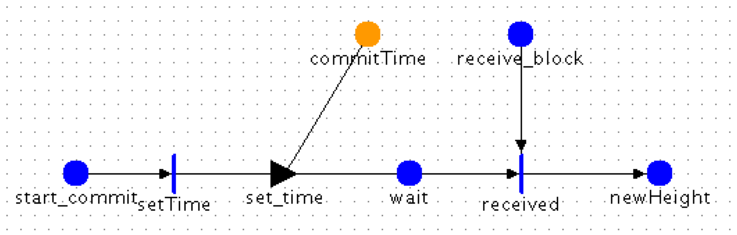

Figure 6: CommitStep atomic SAN

NewHeight Step (Height H)

- Move Precommits to LastCommit and increment height.
- Set StartTime = CommitTime+timeoutCommit
- Wait until StartTime to receive straggler commits
  → goto Propose(H,0)

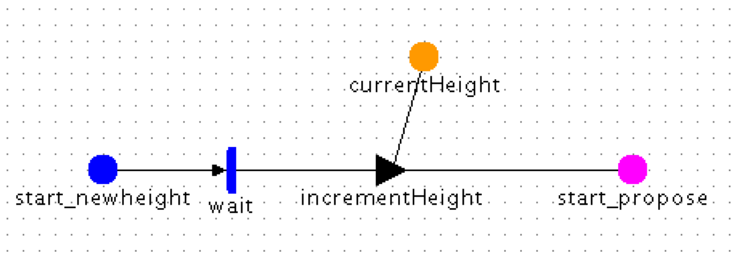

Figure 7: NewHeight atomic SAN prototype

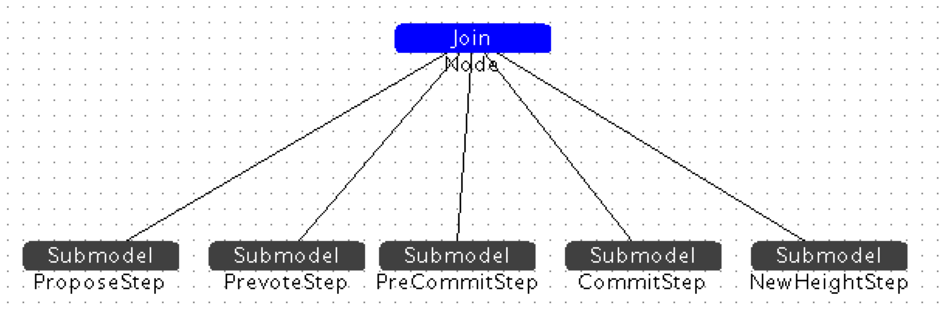

Figure 8: Node Join prototype

5.2. The Modeling Activity

Analogous to the designated coordinator each round of the $\diamond$ algorithm discussed in [7], the Tendermint concensus works with a designated proposer which is chosen by a deterministic and non-choking round robin selection algorithm.[1]. We will try to describe each step in the algorithm as a separate sub-model. A node will be modeled by using the Join facility.

The round-robin algorithm is modeled in a separate model that is joined with the replicated node instances.

6. Conclusion

While I have been able to define a proper formalism and create a skeleton of what the model should look like, it has not yet been fully developed due to the complexity of the multi-agent based concensus algorithm. Therefore, it is not possible to use it in simulation to gain knowledge about its performance at this point in time.
I have experienced Stochastic Activity Networks as a competent extension

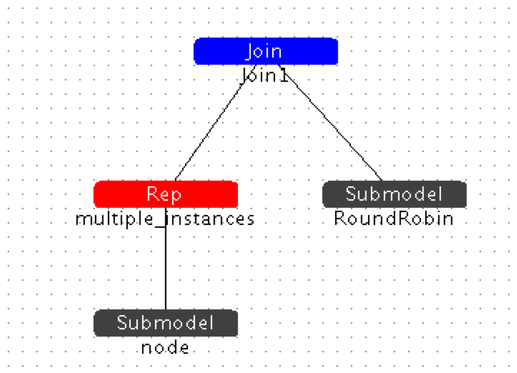

Figure 9: Node Replication prototype

of Stochastic Petri Nets, which enables more complex activity workflows for performance modeling.

Difficulties

While developing the model and submodels, I have experienced some obstacles;
The algorithm description is sometimes more ambiguously phrased as it seems at first sight. This makes it difficult to implement the steps. Reworking the algorithm desciption into more clear pseudo-code might help when further developing the model. There is also not a very clear view on what is communicated between nodes, what takes place at a single node and what certain terms mean exactly.
Besides that, it seemed difficult to access unique (non-shared) instance variables in a join, which would be necessary to couple the round-robin for choosing the designated proposer. This might be due to limitations of the Replicate-Join formalism, or a wrong approach to the problem.

Future work

Naturally, the first work that should be done is to completely finish the model in accordance to the algorithm. When completed, simulations can be executed and compared to real life experiments. Possible extensions at that point would be a more advanced networking submodel to mimic a more realistic network environment.
In the current model structure, there is also no dynamic way of scaling in the amount of node instances, a rework of this structure could be an interesting addition as well.

References

[1] Tendermint Github repository (https://github.com/tendermint/tendermint)

[2] Tendermint ReadTheDocs page(http://tendermint.readthedocs.io/)

[3] Ethan Buchman, *"On the Design and Accountability of Byzantine Fault Tolerant Protocols"*, Tendermint, University of Guelph, Jan 27 2017

[4] Sanders, William H., and John F. Meyer, *"Stochastic Activity Networks: Formal Definitions and Concepts"*, Springer, Berlin, Heidelberg, 2000

[5] A. Schiper et al, *"Performance Analysis of a Consensus Algorithm Combining Stochastic Activity Networks and Measurements"*, Universita di Firenze, 2002

[6] Morteza Golkari & Elham Arshad, *"Stochastic Activity Networks" - presentation*,

[7] Andrea Mario Coccoli, *"On Integrating Modelling and Experiments in Dependability and Performability Evaluation of Distributed Applications."*, PhD thesis, University of Pisa, Italy, 2002

[8] Rajitha Yasaweerasinghelage, Mark Staples and Ingo Weber *"Using architectural modelling and simulation to predict latency of blockchain-based systems"*, University of New South Wales, Australia, 2017

[9] Rajitha Yasaweerasinghelage, Mark Staples and Ingo Weber, *"Predicting Latency of Blockchain-Based Systems Using Architectural Modelling and Simulation"*, University of New South Wales, Australia, 2017

[10] Sukhwani, Harish, et al. *"Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric)"*, In Reliable Distributed Systems (SRDS), 2017 IEEE 36th Symposium on (pp. 253-255). IEEE. 2017

[11] Jae Kwon, *"Tendermint: Consensus without Mining"*, Cornell University, 2014

[12] Möbius Model-Based Environment for Validation of System Reliability, Availability, Security, and Performance, *www.mobius.illinois.edu*, University of Illinois, 2018

[13] Möbius wiki *www.mobius.illinois.edu/wiki*, University of Illinois, 2018