ECE 595: Foundations of Blockchain Systems **December 2, 2020**

Lecture 18

Lecturer: Sreeram Kannan *Scribe: Viswa Virinchi Muppirala*

In this lecture we are going to look deeper at dynamic self allocation where the honest nodes follow a rule and allocate themselves to a shard in order to achieve throughput. We will understand how data availability attacks are handled, we will look at state commitments and how to bootstrap into a new shard fast and also look at how inter-sharding transactions work.

18.1 Dynamic Self-Allocation (continued)

The following notation will help us understanding dynamic self-allocation. Let $\gamma_i(t)$ be the fraction of honest compute power in shard i at time t out of the total power and let $\beta_i(t)$ be the honest compute power in shard i at time t. The total honest power γ will be $\sum_i^k \gamma_i(t)$. The total adversary power β will be $\sum_i^k \beta_i(t)$.
The rate at shard i would be the fraction of honest power,

$$R_i(t) = \frac{\gamma_i(t)}{\gamma_i(t) + \beta_i(t)}$$

Each shard will have its own rate adjustment algorithm where the difficulty of the block is changed according to the rate of block arrival. If many nodes are congregated in a shard, the block mining rate increases and hence the difficulty of mining the block increases. Since the shards have their own rate adjustment algorithm, the rate is given by the fraction of honest power in the shard. The average rate over time T for shard i would be, $\bar{R}_i(T) = \frac{1}{T}\sum_{t=1}^{T} R_i(t)$. We want to maximize the worst shard that has the lowest time average throughput and the adversary's goal is to minimize this. Let $\vec{\gamma(t)} = \{\gamma_i(t)\}_{i=1}^k$ and $\vec{\beta(t)} = \{\beta_i(t)\}_{i=1}^k$. While designing the protocol, honest parties look at past allocations $\{\vec{\gamma(1)} \dots \vec{\gamma(t-1)}\}$, $\{\vec{\beta(1)} \dots \vec{\beta(t-1)}\}$ and output the new allocation $\vec{\gamma(t)}$.

$$\vec{\gamma(t)} = f_t(\vec{\gamma(1)} \dots \vec{\gamma(t-1)}, \vec{\beta(1)} \dots \vec{\beta(t-1)})$$

Note that the adversary knows the protocol design, hence has the knowledge of $f_1 \dots f_T$ functions and decides $\beta_i(t)$ after looking at $\gamma_i(t)$. It minimizes the following quantity

$$\{\vec{\beta(1)}^*, \dots, \vec{\beta(T)}^*\} = \arg\min_{\{\vec{\beta(1)}, \dots, \vec{\beta(T)}\}} \min_i \frac{1}{T}\sum_{t=1}^{T} \frac{\gamma_i(t)}{\gamma_i(t) + \beta_i(t)}$$

The protocol is designed as follows

$$f_1^* \dots f_T^* = \arg\max_{\{f_1 \dots f_T\}} \min_{\{\vec{\beta(1)}, \dots, \vec{\beta(T)}\}} \min_i \frac{1}{T}\sum_{t=1}^{T} \frac{\gamma_i(t)}{\gamma_i(t) + \beta_i(t)}$$

The maximized quantity will be a function of k and is denoted by $\psi(k)$

$$\psi(k) = \max_{\{f_1 \dots f_T\}} \min_{\{\vec{\beta(1)}, \dots, \vec{\beta(T)}\}} \min_i \frac{1}{T}\sum_{t=1}^{T} \frac{\gamma_i(t)}{\gamma_i(t) + \beta_i(t)}$$

To solve this problem, we will look at few examples.

18.1.1 Static uniform allocation

In this case $f_t(.) = 1/k$ and let's assume $\gamma = \beta = 1/2$. The adversary could do a focus attack where it focuses its power onto one shard. $\vec{\beta(t)} = \{\beta, 0, \ldots, 0\}$. In this case $\bar{R}(T) = \{\frac{1}{k+1}, 1, \ldots, 1\}$ and $\psi(k) = \frac{1}{k+1}$.

18.1.2 Mimic the adversary

$\gamma_i(t) = \beta_i(t-1)$. This is self-normalizing as $\gamma = \sum_i^k \gamma_i(t) = \sum_i^k \beta_i(t-1) = \beta$. In this case, if for all t, $\vec{\beta(t)} = \{1/2, 0, \ldots 0\}$ then, $\vec{\gamma(t)} = \{1/2, 0, \ldots 0\}$. Which means $\vec{R(t)} = \{1, 0, \ldots 0\}$. If the honest party had allocated a little to each of the other shards, it could have gained so much rate and we are losing out on a lot of shards in this policy. And ideally we want the summation of $\vec{R(t)}$ to be k.

18.1.3 A mixture of both

Consider the following strategy where $\vec{\gamma(t)} = \frac{1}{2}\beta(\vec{t-1}) + \frac{1}{2}\frac{1}{2k}$. It turns out that $\omega(\frac{1}{\log k}) \leq \psi(k) \leq O(\frac{\log\log k}{\log k})$. $\log\log k$ is essentially a constant and this rate is far better than $\frac{1}{k}$. The analysis behind the result is complicated so we will briefly describe the adversary's attack on the policy. The adversary will slowly concentrate on a shard while the honest power slowly follows the adversary and when the adversary has no more power to allocate to the shard, it will start concentration on a different shard in the same manner creating a cycle.

18.1.4 Using Blackwell approachability [1]

The strategy is as follows, $\gamma_i(t) = C[1/2 = \bar{R}_i(t-1)]^+$ where $1/2$ is the target rate and $\bar{R}_i(t-1)$ is the realized rate. The difference is called backlog. x^+ is defines as $\max(x, 0)$ so if you're achieving more than $1/2$ rate then you don't allocate anything. The constant C is chosen to normalize $\vec{\gamma_i}(t)$. This policy has a striking result where under the worst adversary attack it achieves $1/2$ rate for the worst shard.

$$\psi(k) = \frac{1}{2}$$

Even with an adaptive adversary and adaptive allocation, we are achieving the same results from static adversary and static allocation

18.1.5 Practical implementation of dynamic self-allocation

The honest parties can't practically come to an agreement on which shard they allocate themselves, one way to do this would be flipping a coin based on the $\vec{\gamma(t)}$ distribution and allocate yourself a shard. The second practical constraint is the lack of a clock and synchronization between the parties. All the parties don't have to wake up on the same day and rotate themselves. To tackle this problem, the parties set a timer for a random time that keep them in a shard and when the timer expires they re-allocate themselves according to $\vec{\gamma(t)}$.
This approach can handle heterogeneous sharding and also asynchronous rotation.

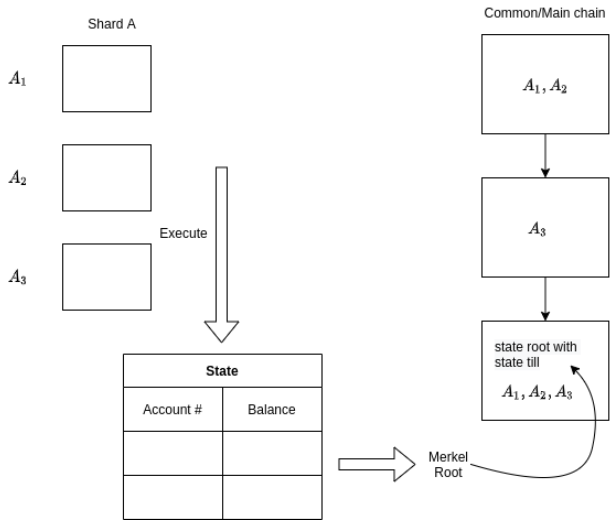

Figure 18.1: The state root is going to be the commitment in blockchain

18.2 Data Availability

We've looked at the data availability issue in the previous lecture where we make sure the all the blocks are available. A simple method to make sure this happens is by flooding the block to everybody which is not communication efficient. A different approach will be as follows. A block producer will create an erasure code for a block which is essentially splitting the data into coded chunks such that a collection of these chunks is sufficient to decode the block. The block producer sends the coded chunks to nodes and each node will get different sets of chunks. If a node receives its corresponding chunks then it is considered that the block is available.

18.3 Bootstrapping

We ask the following question: when a node moves from shard i to shard j, does it need to download the entire ledger? Bootstrapping issues arises because in order to validate a new transaction, the entire ledger needs to be downloaded. We use state commitments to tackle this issue. Consider the following example in figure 18.1 to understand state commitments. Every shard periodically writes its state into the main/common chain (third block in figure 18.1). The state in the figure 18.1 contains the state of shard A till A_1, A_2, A_3. When blocks A_1, A_2 and A_3 from the figure 18.1 are executed or in general all transactions are executed until the end, it creates a data structure with accounts and balances. The Merkle root of this data structure will be the state root that periodically goes into the main chain.
A new node without downloading the whole ledger will just download the state from an another node and verify it using the state root from the main chain. A new problem arises where the nodes from other shards don't know whether the state commitment is valid or not without downloading the blocks of other shards. This is solved by interactive state commitments which we will look at in the next lecture.

18.4 Cross-shard transactions

We will describe cross-shard transactions using the following example. Let shard A have users $\{user1, user2, user3\}$ and let shard B have users $\{user3, user4, user5\}$. Let block A_2 in shard A have the transaction $user3 \rightarrow user5 : \$1$ and block $B3$ in shard B have a transaction that notifies that $user5$ received $\$1$ from shard A which is unverifiable unless the nodes in shard B downloads the blocks from A. To avoid this problem, $user5$ waits for the state commitment from shard A to show up on the main chain and then declares that they received $\$1$ from shard A in shard B which goes into the main chain after the state commitment from shard A. The execution of transactions in shard A will also include the state of $user5$ in the state. The only issues with state commitment is that it introduces latency as nodes wait for periodic state commitments.

References

[1] D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1), 1956.