International Journal of Social Sciences and Management Review

Volume: 05, Issue: 03 " May - June 2022"

ISSN 2582-0176

ON THE RANDOMIZED PROCESS OF TIMESLOT ALLOCATION IN 5IRECHAIN NETWORK

VILMA MATTILA, PRATEEK DWIVEDI, PRATIK GAURI & DHANRAJ DADHICH

5ire (Sustainable Distributed Computing)
Unit Number 101, IFZA Dubai - Building A2, Dubai Silicon Oasis,
United Arab Emirates

https://doi.org/10.37602/IJSSMR.2022.5322

ABSTRACT

In this paper, we study the slot allocation algorithm of 5irechain. This algorithm distributes timeslots to block assemblers. The block as- sembler that receives a certain timeslot gets the opportunity to produce a block in that timeslot. The slot allocation algorithm allocates slots to block assemblers depending upon their weights. That is the block assem- bler with higher weight gets to assemble more blocks than others and thus earns higher incentives.

1.0 INTRODUCTION

Like every cryptocurrency network, 5irechain also requires transactions to be in- cluded into valid blocks and accepted across the network in order to be confirmed. Block production is the most crucial thing for any cryptocurrency network as the stability and consistency of the network depends on it. If the issues related to block production is not properly addressed, it will leave the network in an inconsistent state, and could enable double-spending of tokens. In 5irechain, a block is produced in every 3 seconds unless there is lack of available transactions to construct a block. These blocks are produced by block assemblers who are selected on the basis of their weights that are calculated from multiple pa- rameters. Prospective assemblers offer to assemble blocks in an epoch lasting 48 hours. Then a group of around 50 assembler are chosen depending upon their total weights. The top 50 nodes in terms of total weight are chosen for block assembly. Then these assemblers are allocated timeslots, each of which last 3 seconds. Since, an epoch lasts 48 hours there can be at least 48 1200 or 57600 time slots to be distributed among assemblers. If there are parallel chains, this number will be much higher. The slot allocation algorithm of 5irechain randomly distributes timeslots to assemblers. The algorithm takes a seed to randomize the process of slot allocation, and this random seed is extracted from the 5irechain itself. The slot allocation algorithm ensures that each block assembler will get to assemble a number of blocks in proportion to her own weight. So, a node with a high value of weight will get to assemble a higher number of blocks. So, the nodes will earn more incentives than others with lower weights. The algorithm is decentralized in nature, meaning that all the nodes running the algorithm on similar inputs will produce identical outputs. This will ensure that there will be a general consistency among nodes that are not cut-off from the network.

Volume: 05, Issue: 03 " May - June 2022"

ISSN 2582-0176

2.0 SLOT ALLOCATION METHOD

In this section, we describe how slots are allocated to block assemblers in 5irechain. In the 5ire ecosystem, one epoch lasts for 48 hours. Each epoch is divided into slots of length 3 seconds each. The 5ire ecosystem distributes slots to the assem- blers in a decentralized way. The number of slots in the 5irechain will depend upon the number of nested chains. In each nested chain there will be one slot in parallel to each other. Our scheduling algorithm allocates s slots to all block assemblers across different chains while ensuring that each of the assemblers gets a number of slots in proportion to her total weight. Here, s is the number of slots to be distributed to assemblers. Obviously s is the number of parallel chains in the network. So, there are s time slots to be distributed among n assemblers. Our slot allocation algorithm distributes all the s slots to s block assemblers. The slot allocation algorithm is executed between time tk and tk+1, and it distributes s many time slots of time tk+1. Let there be n ≥ s block assemblers identified as

P1, P2, . . . , Pn. The total weight of Pi is wi, for i [1, n]. The total weights are calculated from multiple factors associated with 5irechain's consensus protocol. The slot allocation algorithm distributes slots in a way so that the assembler wi Pi gets approximatelyn j=1wi wj × 100% of the total slots in a particular epoch. Let us define αi = n j=1 wj . The slot allocation algorithm maintains a variable Ui to store the dynamic number of blocks that have so far been allocated to assembler Pi. Ui is continuously updated as more slots are allocated to Pi. The slot allocation algorithm takes as input a random number β which is extracted from the blockchain itself.

There are two random functions in Algorithm 1. The first function rand1 () takes two integers l , and l . It samples a random r ←−$ [l , l], and returns it.

The other function rand() takes as input a seed β , and two fractional numbers µ1, µ2 ∈ [0, 1], such that µ1 < µ2, and return a random fractional number ν satisfying µ1 ν µ2. The two random functions are deterministic in nature. This ensures that they output the same values in all the nodes. The seed β is a number extracted from the 5irechain itself. It is the hash of all the blocks across all parallel chains that occurred exactly hundred time slots back in the

5ire ecosystem. If there were multiple chains then the blocks occurring on all the chains exactly hundred slots back need to be considered as shown in Figure 1. The use of the random seed β ensures that the slot allocation schedule cannot be predicted too early. The algorithm first calculates the fraction of slots allocated to each assembler. This is denoted by ai in Algorithm 1. Then the algorithm calculates the difference bi between αi and ai. If the difference, is negative then it means that the assembler i has so far been allocated more slots than what she deserves depending upon her weight. So, the algorithm does not allocate slots to that assembler in the current time slot. Instead, all the assemblers for whom the

---

Require: $\alpha_i, U_i : i \in [1, n]$, slot no. $=s, \beta$

Ensure: Γ, U_i

Volume: 05, Issue: 03 " May - June 2022"

ISSN 2582-0176

for $i = 1$ n **do**

υ_i

$\alpha_i = {}_{n}\Sigma$

$_{j=1}\,\upsilon_j$

$b_i = \alpha_i - a_i$

if $b_i < 0$ **then**

$b_i = 0$

end if

end for

$\Gamma = \emptyset, \zeta = \emptyset, t = 1$

while $|\zeta| < s$ **do**

$c = 0$

$L = 0$

$B = D = \emptyset$

for $k = 1 \rightarrow n$ **doif** k $\in \zeta$ **then**

continue

else

$c = c + b_k L$
$= L + 1 B[L]$
$= k$

$D[L] = c$

end if

end for

if $c = 0$ **then**

$z = rand1(1, L)\Gamma$
$[t] = B[z]$

$U_z = U_z + 1$

else

$m = rand(\beta, 0, c)$

for $i = 1 \rightarrow L$ **do**

if $m < D[i]$ **then**

$\Gamma[t] = B[i]$

$\zeta = \zeta \cup \{B[i]\}$

Volume: 05, Issue: 03 " May - June 2022"

ISSN 2582-0176

$U_i = U_i + 1$

end if

end for

end if

$t = t + 1$

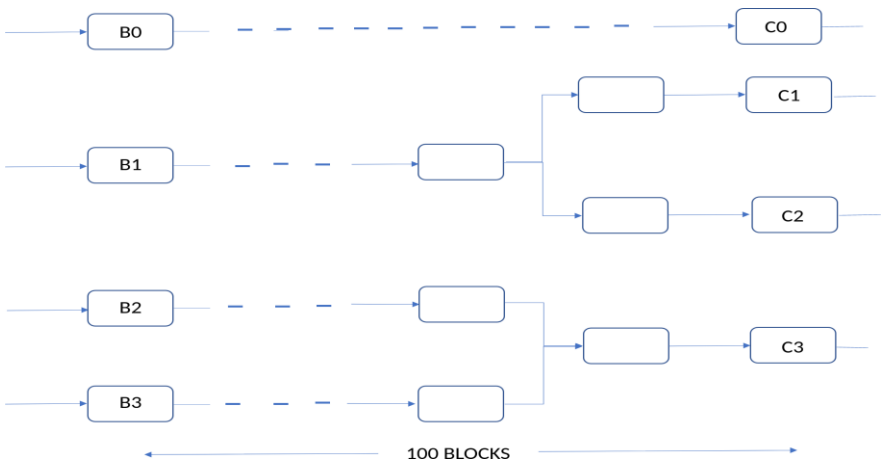
end while

difference is positive are chosen. If the difference is not positive for any of the assemblers, then the slots are randomly allocated to the block assemblers. For example, when the fraction of allocated slots are same as the fraction of weights,

i.e. when $a_i = \alpha_i$, for all $i \in [1, n]$, then the slots will be randomly distributed to the assemblers, with the condition that an assembler must not get more than one concurrent slots across multiple shards. So, in a particular time, a block assembler does not get to assemble more than one slot if there multiple chains existing. If the difference $b_i = \alpha_i - a_i$ is higher than zero for some assemblers, then the algorithm randomly distributes the slots on the basis of b_i's for all $i \in [1, n]$, such that $b_i > 0$. The algorithm stores the set of assemblers that have been allocated slots in the dynamic variable ζ. At any point during the execution of the algorithm, there are $s - |\zeta|$ slots to be allocated to $s - |\zeta|$ assemblers. Assume that $\Psi = \{i : b_i > 0, i \in [1, n]\} \setminus \zeta$. That is, Ψ is the set of indices of assemblers that have not been allocated any block in the next slot, and for whom b_i is positive. Our algorithm allocates a time to a an assembler in Ψ with probability as follows:

$$p_i = \frac{b_i}{\sum_{j \in \Psi} b_j}$$

Hence, $\sum_{i \in \Psi} p_i = 1$. Hence, the $|\zeta|+1$'th slot is assigned to one of the assemblers in Ψ with probability 1. This way all the s slots in time t_{k+1} is allocated to s out of n assemblers.



Volume: 05, Issue: 03 " May - June 2022"

ISSN 2582-0176

Fig. 1. Extracting randomness from the 5irechain

2.1 Extracting randomness from 5irechain

The slot allocation algorithm of 5irechain distributes time slots to block assem- blers randomly while ensuring that the number of slots received by an assembler is in proportion to her total weight. At the same time, we have to ensure that the slot allocation cannot be predicted too early. For that reason, we need to incorporate a randomness into the slot allocation process such that it is hard to predict the randomness before it is actually generated. We observe that the 5irechain network itself can be used to produce the randomness. The blocks in 5irechain network can be treated as a source of randomness. In our slot allo- cation algorithm, we use a randomness called β to randomly distribute slots. While distributing the slots for a particular time, we extract randomness from the blocks that occurred on the 5irechain exactly 100 positions back. The process is depicted in Figure 1. In this figure, the most recent blocks are C0, C1, C2, and C3. Obviously, there are four parallel chains or shards. We need to distribute next four slots across four parallel chains. While distributing the four slots, we extract a randomness β from the blocks that occurred 100 positions back. As can be observed in the figure, there were five blocks across five shards that oc- curred in the network 100 positions back. They are denoted in the figure as B0, B1, B2 and B3. So, we computes β as the hash of those blocks, that is, β = Hash(B0, B1, B2, B3). Thus the randomness β needs to be calculated from all the blocks across all the parallel chains that existed exactly 100 positions back even when some of them are terminated later or when new chains are created from older chains. In Figure 1, it can be observed that two chains are merging into one, whereas one chain is getting split into two. Thus, creation of newer chains or deletion of older chains does not have any impact on the way β is com- puted. The value of β only depends on the blocks that occurred 100 positions back. Since, in 5irechain, the duration of a slot is 3 sec, β cannot be computed more than 300 seconds ago. Note that in case of multiple parallel chains, not all chains must necessarily have the hundredth block at the same time slot. This is due to the fact that one block in a particular time slot on a particular chain might be missing. It is quite possible than a block in a particular time slot does get assembled or attested.

Theorem 1. Our 5irechain slot allocation algorithm allocates blocks to assem- blers in proportion to their weights.

Proof. Let us assume that $U_{ij} : i \in [1, n]$ is the total number of slots allocated to P_i after timeslot j. Also assume that there are τ many sequential slots. Also assume that a particular epoch consists of μ slots. If there are no nested chains in the entire epoch lasting 48 hours, μ will be equal to τ. As such $\mu = \sum_{i=1}^{n} U_{i\tau}$. We assume that $\alpha_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$, where w_is are the weight of P_i. We also assume that $\alpha_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$, where w_is are the weight of P_i. Let us assume that $U_{ij}/\sum_{k=1}^{n} U_{kj} > \alpha_i$, but $U_{i(j-1)}/\sum_{k=1}^{n} U_{k(j-1)} < \alpha_i$ for some $i \in [1, n]$. Let us assume that the number of parallel chains in timeslot j is r. The slot allocation algorithm ensures that if there are multiple chains

Volume: 05, Issue: 03 " May - June 2022"

ISSN 2582-0176

in a timeslot, a block assembler will not get more than one slot during that timeslot. So, $U_{ij} - U_{i(j-1)} = 1$ holds. Also, $\sum_{k=1}^{n}(U_{kj} - U_{k(j-1)}) \geq 1$. Hence, $\frac{U_{ij}}{\sum_{k=1}^{n} U_{kj}} \leq \frac{U_{i(j-1)}+1}{\sum_{k=1}^{n} U_{k(j-1)}+1}$. Now, $U_{i(j-1)} < \alpha_i * \sum_{k=1}^{n} U_{k(j-1)}$. So, $\frac{U_{ij}}{\sum_{k=1}^{n} U_{kj}} < \frac{\alpha_i * \sum_{k=1}^{n} U_{k(j-1)}+1}{\sum_{k=1}^{n} U_{k(j-1)}+1} = \alpha_i + \frac{1-\alpha_i}{\sum_{k=1}^{n} U_{k(j-1)}+1}$. Thus, $|\frac{U_{ij}}{\sum_{k=1}^{n} U_{kj}} - \alpha_i| < \frac{1}{\sum_{k=1}^{n} U_{k(j-1)}+1}$. From this we get, $|\frac{U_{i\tau}}{\sum_{k=1}^{n} U_{k\tau}} - \alpha_i| < \frac{1}{48\times1200} \approx 0.0000173$. From the above result, it is evident that for any assembler $P_i : i \in [1, n]$, $U_{i\tau}/\mu < \alpha_i + 0.0000173$. We shall now try to obtain a lower bound on $U_{i\tau}/\mu$, for an $i \in [1, n]$. $\sum_{i=1}^{n} U_{i\tau}/\mu = 1$. So, for any $i \in [1, n]$, $U_{i\tau}/\mu = 1 - \sum_{j=1, j\neq i}^{n} U_{j\tau}/\mu$. Since, $U_{j\tau}/\mu < \alpha_j + 0.0000173$, $U_{i\tau}/\mu > 1 - \sum_{j=1, j\neq i}^{n} \alpha_j - (n-1)0.0000173 = \alpha_i - (n-1) * 0.0000173$. If $n = 50$, $U_{i\tau}/\mu > \alpha_i - 0.0000173 * 50$. Thus, $\alpha_i - 0.0000173 < U_{i\tau}/\mu < \alpha_i + 0.00008680555$.

3.0 CONCLUSION

In this paper, we study the slot allocation algorithm of 5irechain. The algorithm allocates timeslots to block assemblers randomly in a completely decentralized fashion. Every block assembler gets a number of slots in proportion to her own weight as calculated before the commencement of an epoch. The algorithm makes use of two random number generators that take as input a random seed generated from the 5irechain itself.

REFERENCES

Zheng Z, Xie S, Dai H, Chen X, Wang H, An overview of blockchain technology: architecture, consensus, and future trends. In: Proceedings of the 2017 IEEE international congress on big data (BigData congress). IEEE, pp 557–564, 2017.

Matching with indifferences: A comparison of algorithms in the context of course allocation European Journal of Operational Research, 2017.

C. C. Han, K. J. Lin, and C. J. Hou. Distance-constrained scheduling and its applications to real-time systems. IEEE Trans. on Computers, 45(7):814-826, 1996.

D. Puthal, S. P. Mohanty, P. Nanda, E. Kougianos and G. Das, "Proof-of-authentication for scalable blockchain in resource-constrained distributed systems", Proc. IEEE Int. Conf. Consum. Electron, 2019.

Q. Wang, B. Ye, T. Xu and S. Lu, "An approximate truthfulness motivated spectrum auction for dynamic spectrum access", Proc. IEEE Wirel. Commun. Netw. Conf., pp. 257-262, 2011.

R. Carr and S. Vempala, "Randomized metarounding", Random Struct. Alogrithm, vol. 20, no. 3, pp. 343-352, 2002.

S.-H. Wie and D.-H. Cho, "Time slot allocation scheme based on a region division in CDMA/TDD systems", IEEE Veh. Technol. Conf., vol. 4, pp. 2445-2449, 2001.

Volume: 05, Issue: 03 " May - June 2022"

ISSN 2582-0176

H. Chung, M. Kim, N. Kim and S. Yun, "Time slot allocation based on region and time partitioning for dynamic TDD-OFDM systems", Proc. IEEE Veh. Technol. Conf., 2006.

