UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ (UNIOESTE)
CENTRO DE ENGENHARIA E CIÊNCIAS EXATAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GABRIEL FERREIRA DOS SANTOS

SIMULAÇÃO DE STAKE E SELEÇÃO DE VALIDADORES EM RUST: DESENVOLVIMENTO DE UM SIMULADOR EDUCACIONAL BASEADO EM PROOF OF STAKE

Foz do Iguaçu
2024

GABRIEL FERREIRA DOS SANTOS

Simulação de stake e seleção de validadores em rust: desenvolvimento de um simulador educacional com base em proof of stake

Trabalho de pesquisa desenvolvido com o objetivo de aprofundar os conhecimentos em Blockchain e Web3, por meio da implementação de um simulador de seleção de validadores baseado no modelo Proof of Stake, utilizando a linguagem de programação Rust e inspirado nos princípios da rede Polkadot.

Foz do Iguaçu

2024

SUMÁRIO

1. INTRODUÇÃO

Com a evolução da internet e o surgimento de protocolos mais sofisticados, que permitem não só a leitura, mas também a escritura de dados, surgiu a Web3, que introduziu o sistema de economia virtual descentralizada. Neste contexto, a Blockchain, uma estrutura de blocos de dados interligados por criptografia, garante a segurança e a imutabilidade das transações realizadas na rede. Esse sistema descentralizado assegura que transações e trocas sejam executadas de forma confiável e com maior liberdade para os usuários.

Para a confiabilidade e segurança dessas interações, foram criados Contratos Inteligentes (Smart Contracts). Estes contratos são implementados por chaves públicas e privadas, que gerenciam a assinatura digital das operações e executam-as automaticamente quando determinadas condições são atendidas. Conhecida pelo alto desempenho e segurança de memória, a linguagem de programação Rust vem se tornando cada vez mais popular para aplicações e desenvolvimento de sistemas Web3, como é o caso das plataformas baseadas em Blockchain.

Além disso, por ser uma rede descentralizada, computadores pessoais e/ou de alto nível contribuem para o seu funcionamento. Em contrapartida, os usuários são recompensados com tokens, moeda virtual que alimenta o mercado virtual e são utilizadas em transações. Para incentivar essa participação direta na rede, a Blockchain adota um sistema que prova sua contribuição. No caso do Bitcoin, a prova é realizada por meio da Proof of Work (PoW), onde a recompensa é dada com base no cálculo de algoritmos complexos feitos pela máquina do usuário, popularmente conhecida como “mineração”. Já na Polkadot, o sistema utiliza da Proof of Stake (PoS), onde validadores são selecionados com base em tokens travados (Stakes), e, ao validá-los, recebem recompensas. Caso o validador produza um bloco defeituoso, ele será penalizado.

Baseado no sistema da Polkadot, foi implementado um programa, utilizando a linguagem Rust e suas bibliotecas externas, para simulação da seleção de validadores utilizando a quantidade de stakes que cada usuário possui. A seleção ocorre a cada seis rodadas. A partir do retorno mostrado no terminal, gerado pelo código, é extraído as informações de cada seleção e as recompensas ou penalidades que aplicadas ao validador. O sistema foi inspirado no modelo PoS, a

fim de auxiliar aqueles que querem ingressar no ambiente Web3 utilizando DOT como token e demonstrar como a linguagem utilizada opera.

2. FUNDAMENTOS TEÓRICOS

O simulador consiste em fundamentos principais da base da Web3. Entender melhor esses conceitos é essencial para o aprofundamento do projeto. A seguir, será descrito em detalhes cada fundamento utilizado na construção do simulador.

2.1 BLOCKCHAIN

A Blockchain surgiu como uma resposta direta à necessidade de sistemas digitais mais confiáveis, transparentes e descentralizados. Projetada inicialmente como base para o funcionamento do Bitcoin, a Blockchain permite que os dados — como as transações — sejam armazenados em blocos encadeados de forma imutável.

Cada bloco é vinculado ao anterior através de funções criptográficas de hash, garantindo a segurança e a integridade da informação. Caso qualquer dado de um bloco seja alterado, seu hash também será modificado, invalidando o bloco e todos os subsequentes da cadeia.

Diferentemente dos Bancos de Dados, a blockchain não depende de um servidor centralizado para armazenar as informações. Em vez disso, funciona em uma rede peer-to-peer, que replica os dados entre os diversos nós conectados. A integridade da cadeia é mantida por algoritmos de consenso, os quais garantem que todas as cópias da rede concordem sobre o estado atual do sistema. No caso da Polkadot, o algoritmo utilizado para validar os blocos é uma variação da Proof-of-Stake.

2.2 PROOF-OF-STAKE (PoS)

O PoS é um mecanismo de consenso da blockchain que seleciona quem pode validar blocos com base na quantidade de criptomoedas que as pessoas possuem e bloqueiam (“faz stake”) na rede. É uma alternativa ao PoW, usado no Bitcoin, e foi criado para ser mais eficiente e escalável, com baixo custo de energia.

Os usuários que possuem mais stakes, têm maior chance de ser selecionado pelo algoritmo para validar o próximo bloco e receber recompensa. Caso o validador haja de forma maliciosa ou errada, como fraudar blocos ou ficar muito tempo offline, parte do seu stake é retirado ou destruído como penalidade (slashing).

3. METODOLOGIA

3.1 ETAPA DA CONSTRUÇÃO LÓGICA

No estágio inicial, foi realizado um estudo aprofundado na linguagem de programação Rust, com base no livro *The Rust Programming Language*, para a

aplicação no simulador. A lógica e as estruturas que seriam utilizadas no desenvolvimento do programa foram previamente escritas no papel, contemplando:

- Criação de usuários, com perfis de nível de conhecimento (iniciante, intermediário, avançado e profissional).
- Níveis de conhecimento, definindo a quantidade aleatória de stakes que cada usuário fará, aplicando um percentual para cada perfil, simulando a ação de cada usuário.
- Inscrição para validador, somente para perfil avançado e profissional e atribuição de cargo “Inscrito”.
- Algoritmo de seleção de validadores e atribuição de cargo “Validador”.
- Algoritmo de recompensa/penalidade, simulando validação de blocos.
- Sistema de rodadas, sendo 360 no total. A cada seis rodadas, o validador é retirado e efetua-se uma nova seleção.
  O objetivo é organizar as ideias e preparar a lógica para a construção de funções, estruturas de dados e implementações necessárias para simular um PoS.

3.2 FERRAMENTAS

Após a etapa da construção lógica do programa, foi definido o ambiente que seria utilizado para sua definição. O *Visual Studio Code*, da *Microsoft*, apesar de ser um editor de código-fonte, fornece suporte suficiente para a linguagem Rust. Extensões, como *rust-analyzer* e *Rust Syntax*, foram instaladas para auxiliar no desenvolvimento do código. Além disso, a linguagem foi instalada localmente, uma vez que o sistema operacional não apresenta os pacotes por padrão. Arquivos foram criados para organizar o trabalho em partes: main.rs (principal), `user.rs` e `validator.rs`.

4. DESENVOLVIMENTO

Com a lógica desenvolvida e as ferramentas preparadas para o uso, foi possível dar início ao desenvolvimento do simulador. As implementações foram organizadas em arquivos específicos: `user.rs` para funções associadas e métodos relacionados aos usuários, enquanto `validator.rs` abrange a lógica estrutural dos validadores. Todos os módulos serão adicionados ao arquivo principal para organizar o registro dos usuários, elaboração do sistema de rodadas e geração de uma interface simples no terminal.

4.1 CRIAÇÃO DE USUÁRIOS E ATRIBUIÇÃO DE STAKES

Primeiramente, dentro do arquivo user.rs, fez-se necessário declarar um novo tipo estruturado (`struct`) chamado User, com dados importantes para o registro, como nome, saldo, validador, stake e perfil. Em seguida, foi definido um tipo de dado enumerado (`enum`) `PerfilStake` para compor os níveis de conhecimento a serem

escolhidos pelo usuário. No arquivo `validator.rs`, projetou-se um enum Validador, que armazena os diferentes estados: `NaoValidador`, `Inscrito` e `Validador`. O Validador então foi passado como crate para o arquivo de usuário.

Prosseguindo com o processo, o tipo User foi modelado com o comando `impl`, que permite associar funções e estabelecer métodos à estrutura.

Para a execução do processo de criação, foi projetada uma função pública `novo()`, que recebe como parâmetros uma string para nome, `PerfilStake` e Validador, além de atribuir valores para cada informação do dado estruturado, criando e retornando uma instância. De forma adicional, é alocado mil tokens por padrão à informação de saldo. Logo em seguida, é implementada a função `realizar_stake()`, que retorna um valor inteiro como parâmetro e o retira do saldo para alocar em stakes, caso esse saldo seja maior que o valor. Essa nova função será utilizada em outra, chamada `aplicar_stake_profile()`, onde o percentual de cada perfil será utilizado para retornar um valor aleatório dentro de um limite estabelecido para atribuir stake. Cada perfil contém uma porcentagem: Iniciante (0-25%), Intermediário (25-50%), Avançado (50-75%) e Profissional (75-100%).

Esses novos mecanismos permitem a definição do usuário com todos os dados necessários para o funcionamento do sistema. Porém, o simulador requer algoritmos de sorteio e de recompensas que ainda não foram desenvolvidos.

4.2 INSCRIÇÃO, SELEÇÃO E RECOMPENSA DE VALIDADORES

Para desenvolver o algoritmo da seleção de validadores e o sistema de recompensa/penalidade, alguns passos são necessários. De início, foi formulada uma função `inscrever_user_val()` com o intuito de disponibilizar a opção para o usuário se inscrever como validador. Caso sua escolha seja afirmativa, receberá o status de “inscrito”. Senão, receberá “`NaoValidador`”. Isso separa quais usuários possuem a possibilidade de serem selecionados pelo algoritmo.

Em extensão à etapa anterior, implementa-se `executar_sorteio()`. O objetivo é criar um vetor que servirá como roleta, filtrar os usuários que possuem o status “`Inscrito`”, determinar a quantidade de entradas com base em seu stake e inseri-los. Assim, o algoritmo escolhe, pseudo aleatoriamente, um novo validador. Se não houver inscritos, o programa não selecionará ninguém e retornará a mensagem “Nenhum validador elegível para o sorteio”.

Depois de desenvolver o selecionador de validador, o sistema precisa alocar recompensa ou penalidade ao escolhido. É projetada, então, a função `recompensa_penalidade_validadores()`, na qual uma variável chance recebe um valor aleatório entre 0 e 100, recebe o validador e, se o valor for igual ou superior a 85, recebe uma recompensa de 0,75% do valor de seu stake atual. Se for acima, o selecionado perde 0,5%.

Por fim, `remover_validador_do_inscrito()` foi criada para remover o

status de Validador e voltar ao estado Inscrito, possibilitando uma nova seleção de validadores.

4.3 INTERFACE DE USUÁRIO E SIMULADOR DE SELEÇÃO

O sistema se encontra parcialmente concluído. O simulador ainda necessita de uma interface na qual o usuário pode registrar seu nome, escolher o perfil que mais se identifica e optar por se inscrever como validador.

Dentro de `main.rs`, foi criado um vetor que armazenará os usuários criados. Logo após, é desenvolvida a função `inscrever_user()`, responsável por receber o nome e as opções desejadas.

Primeiramente, solicita-se que o usuário informe seu nome. O valor digitado no terminal é recebido pelo programa e guardado na memória. Em seguida, é requisitado o perfil com o qual mais se identifica, apresentando-se as opções de 1 a 4, conforme os perfis predefinidos. Caso a escolha seja Avançado ou Profissional, o sistema disponibiliza a opção de inscrição para validador.

Depois que o sistema recebe todos os valores, é fornecido como argumento para a função `criar_user()`, que atribui cada valor à sua respectiva área da estrutura User e aloca no vetor, emitindo no terminal a mensagem “Usuário criado com sucesso”.

Por fim, na função `main()`, foi declarada um laço de repetição que itera de 0 a 10 com o intuito de criar dez usuários por meio da chamada à função `inscrever_user()`. Em seguida, inicia-se estruturas de repetições aninhadas, onde a primeira faz uma iteração de 0 a 60, executa a seleção utilizando a função `executar_sorteio()`, enquanto o segundo laço percorre de 0 a 6 atribuindo o sistema de recompensa com a função `recompensa_penalidade_validadores()`. Em seguida, após o fechamento da segunda iteração, realiza-se a remoção dos status por meio da função `remover_validador_do_inscrito()`, permitindo que o processo seja reiniciado.

Com isso, conclui-se a etapa de desenvolvimento do simulador. Próximo passo é iniciar o programa, retirar os resultados que gerar e aplicá-los à uma tabela, para extrair as estatísticas.

5. RESULTADOS

Ao inicializar o programa, o simulado solicita o nome de usuário. Neste primeiro teste, foram realizados os registros de dez pessoas fictícias: Gabriel, Amanda, Silvio, Ana, Maria, Julia, Fabiana, Mariana, João, e Marcos. Cada um com escolhas diferentes para perfil e validador. O sistema rapidamente atribuiu stakes automáticos, respondendo corretamente à quantidade aproximada que cada nível de conhecimento possui. Assim, cada usuário criado possuía uma quantidade diferente no saldo e na stake, permitindo uma simulação mais fiel à realidade.

Abaixo segue uma análise detalhada da simulação:

Nome	Perfil	Validador?	Stake inicial	Saldo inicial
Gabriel	Avançado	Sim	601	399
Amanda	Profissional	Sim	905	95
Silvio	Avançado	Sim	653	347
Ana	Profissional	Sim	764	236
Maria	Avançado	Não	699	301
Julia	Intermediário	Não	411	589
Fabiana	Iniciante	Não	4	996
Mariana	Avançado	Não	698	302
João	Avançado	Sim	575	425
Marcos	Profissional	Sim	785	215

Nota-se que Amanda foi o usuário que mais travou tokens nesse teste. Marcos e Ana estão logo atrás, com médias altas. Fabiana travou uma quantidade muito abaixo do esperado, simulando uma pessoa com baixa experiência no sistema.

Aqueles que optaram por ser um validador foram escolhidos pelo algoritmo e receberam recompensas e/ou penalidades. Segue a tabela do resultado, em ordem decrescente, gerado pelas rodadas de seleção de validadores:

Nome	Stake Final	Vezes selecionado	Ganhos	Perdas	Variação Total
Amanda	1596	17	+765	-74	+405
Ana	1371	16	+640	-33	+373
Marcos	1053	9	+296	-28	+195
Silvio	870	7	+217	-0	+116
Gabriel	718	5	+122	-5	+117
João	667	6	+112	-20	+41

Percebe-se que, de fato, o usuário detentor do maior stake será escolhido mais vezes, portanto, terá mais chances de receber recompensa, do mesmo modo a

penalidade também. Amanda, Ana e Marcos foram os usuários que mais travaram tokens no início. O algoritmo de seleção atendeu o viés proporcional ao valor congelado, variando de forma coerente com o nível de conhecimento de cada um.

Além disso, nota-se que as penalizações são proporcionais à frequência de seleção, mas com menor impacto geral. Por exemplo: Amanda foi penalizada e perdeu 74 stakes, mas em compensação recebeu 405 a mais. Silvio não foi penalizado. A análise obtida foi que, apesar das perdas, a recompensa é maior, tornando atrativo o sistema, bem como evidencia o funcionamento caso haja falha na validação de um bloco.

O valor da recompensa pode não ser linear, isto é, usuários com valores próximos, mas com número de seleções diferentes, podem apresentar quase o mesmo ganho. É o caso dos validadores Gabriel e Silvio, que possuem stake próximos de 600, foram selecionados 5 e 7 vezes, respectivamente, e a variação total foi de 117 e 116.

À medida que os validadores são selecionados, a quantidade de token aumenta no sistema. Isso representa um crescimento que, a longo prazo, pode gerar uma inflação, desvalorizando a criptomoeda. Medidas devem ser implementadas para controlar essa emissão, como taxas de transição, slashing efetivo, etc. A penalidade, no simulador, não identifica qual tipo de erro foi detectado na validação. Esses são mecanismos que podem ser adicionados futuramente ao programa.

6. CONCLUSÃO

O simulador foi criado com intuito de testar e estudar, de forma didática, o funcionamento da seleção de validadores de blocos com base no mecanismo Proof-of-Stake. A linguagem Rust, utilizada para implementar o programa, permite maior segurança e eficiência na questão da memória. Com os resultados obtidos da simulação, foi possível identificar que o algoritmo atua com performance no quesito criar usuários, atribuir stake automático, sortear os validadores com base nos tokens travados e atribuir recompensas e penalidades, de forma pseudo aleatória, que garante um realismo estatístico nas observações. Apesar de acessível e direto, o código permite escalabilidade, assim como a melhoria da legibilidade e do controle da inflação. Conclui-se que o mecanismo de consenso trabalha de forma eficiente para o sistema da Blockchain, inferindo na funcionalidade das transações e da segurança do mercado virtual, fornecendo uma solução viável para algoritmos de alto gasto energético, como PoW. O simulador entrega o esperado.

REFERÊNCIAS

- KLABNIK, Steve. The Rust Programming Language. **The Rust Programming Language,** 2025. Disponível em: https://doc.rust-lang.org/book/. Acesso em: 02 maio 2025.
- CHUI, Michael; BYRNE, Robert; NADEAU , Marie-claude; ROBERTS, Roger; HAZAN, Eric. What is Web3?. **What is Web3 technology (and why is it important)? | McKinsey,** 2023. Disponível em: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-web3. Acesso em: 02 maio 2025.
- WEB3 FOUNDATION. *Polkadot Developer Docs: Develop*. Zug: Web3 Foundation, 2024. Disponível em: https://docs.polkadot.com/develop/. Acesso em: 03 maio 2025.