Simulando o Hedera Hashgraph

Guilherme de L. Ferreira[1], **Pedro H. F. Von Zuben**[1],
Raphael de C. Mesquita[1], **Yuri V. C. de M. A. da Silveira**[1]

[1]Instituto de Computação – Universidade Federal do Rio de Janeiro (UFRJ)

DRE: 121143034, DRE: 119055699, DRE: 118020104, DRE: 119104804

Resumo. *O artigo "Simulando o Hedera Hashgraph"explora o hashgraph, uma tecnologia de distributed ledger que supera as limitações das blockchains tradicionais. O hashgraph oferece alta segurança, escalabilidade e eficiência utilizando protocolos como "Gossip about Gossip" e "Virtual Voting". Este estudo detalha a estrutura e funcionamento do hashgraph, incluindo a propagação de informações e o processo de consenso, além de comparar suas vantagens em relação às blockchains. A implementação demonstra como o hashgraph se comporta na questão de escalabilidade de usuários na rede.*

Abstract. *The article "Simulating the Hedera Hashgraph"explores hashgraph, a distributed ledger technology that overcomes the limitations of traditional blockchains. Hashgraph offers high security, scalability, and efficiency using protocols such as "Gossip about Gossip"and "Virtual Voting."This study details the structure and operation of hashgraph, including information propagation and the consensus process, and compares its advantages to blockchains. The implementation demonstrates how hashgraph behave regarding user escalability on the network.*

1. Introdução

1.1. *Ledger*

Um *ledger* é um registro que documenta todas as transações de um sistema, guardando informações relevantes sobre cada um deles. Embora seu conceito possa ser aplicado a qualquer tipo de registro organizado, seu uso mais comum é para registrar transações financeiras de uma empresa (sendo chamado, nesse caso, de livro-razão).

1.2. *Distributed Ledger Tecnology (DLT)*

Uma *distributed ledger* é uma versão avançada do *ledger* tradicional, com algumas características-chave:

- **Descentralização:** Em vez de um único ponto central onde todas as transações são registradas, os registros são distribuídos entre múltiplos nós (computadores) em uma rede. Isso significa que não há uma autoridade central controlando o *ledger*.
- **Consistência e Confiabilidade:** Como os registros são distribuídos entre muitos nós, todos os nós mantêm uma cópia do *ledger* e devem concordar sobre o estado das transações. Isso torna o sistema mais resistente a falhas e manipulações.

- **Imutabilidade:** Uma vez que uma transação é registrada em um *distributed ledger*, ela é geralmente imutável. Qualquer tentativa de alterar uma transação registrada é facilmente detectada porque todas as cópias do *ledger* em todos os nós da rede devem concordar sobre o estado das transações.

1.3. *Blockchain*

A *Blockchain* é um tipo específico de *distributed ledger* que ganhou uma atenção considerável nos últimos anos devido a características únicas como: descentralização, imutabilidade, anonimidade, segurança e auditabilidade. Ela é amplamente implementada em criptomoedas e diversas aplicações não monetárias.

Em uma *Blockchain*, os registros de transações entre pares de uma rede *P2P* (*peer-to-peer*) são armazenados em um *ledger*. Cada participante da rede recebe uma cópia do *ledger*, garantindo a auditabilidade e a descentralização. As transações são agrupadas em blocos, e cada bloco contém uma chave *hash* que os vinculam ao bloco anterior. Isso torna extremamente difícil alterar registros anteriores sem ser detectado, criando uma cadeia de blocos imutável.

A criação de um novo bloco segue regras definidas por algoritmos de consenso, como Prova de Trabalho (*PoW*) ou Prova de Participação (*PoS*). Esses algoritmos garantem que todas as partes concordem sobre a validade das novas transações e a ordem em que são registradas.

A segurança na *Blockchain* é garantida pela utilização de criptografia avançada, que protege a integridade e autenticidade das transações. No entanto, a segurança ainda pode ser comprometida por ataques como o *"51% attack"* em *Blockchains* baseadas em *PoW* e o *"nothing-at-strike"* em *Blockchains* baseadas em *PoS*.

Finalmente, a escalabilidade também é um grande desafio para a *Blockchain*, referindo-se à sua capacidade de processar um grande volume de transações de forma rápida e eficiente à medida que a rede cresce. A arquitetura atual de algumas *Blockchains* pode limitar a velocidade de transação e aumentar os custos operacionais. Soluções emergentes como *sharding*, *sidechains* e redes de segunda camada estão sendo desenvolvidas para melhorar a escalabilidade e eficiência da *Blockchain*, possibilitando maior adoção e uso em escala global.

1.4. *Hashgraph*

O *hashgraph*, proposto pela *Hedera Hashgraph*, é um outro tipo específico de *distributed ledger* que visa superar algumas limitações da *Blockchain* tradicional, oferecendo melhorias em vários aspectos importantes:

- **Segurança:** A segurança no *Hashgraph* é garantida pela resistência a falhas bizantinas assíncronas (*aBFT*), o que significa que ele pode manter o funcionamento e a integridade mesmo na presença de atores mal-intencionados. Esta característica torna o *Hashgraph* robusto contra uma variedade de ataques, incluindo aqueles que visam comprometer a rede através da manipulação de um número significativo de nós. Comparativamente, *Blockchains* tradicionais, como o *Bitcoin*, usam a Prova de Trabalho (*PoW*) para segurança, mas isso os torna vulneráveis ao *"51% attack"*.

- **Escalabilidade:** Um dos principais problemas das *Blockchains* tradicionais é a escalabilidade. As *Blockchains* podem enfrentar dificuldades em processar um grande volume de transações rapidamente à medida que a rede cresce. O *Hashgraph*, por outro lado, é projetado para ser altamente escalável, permitindo milhares de transações por segundo (*TPS*). Isso é possível devido à sua estrutura de grafo acíclico dirigido (*DAG*), que permite que as transações sejam processadas em paralelo, ao invés de sequencialmente como nas *Blockchains*.
- **Regras de Consenso:** O *Hashgraph* utiliza um método de consenso diferente das *Blockchains* tradicionais. Em vez de depender de algoritmos como Prova de Trabalho (*PoW*) ou Prova de Participação (*PoS*), ele usa a combinação de *"Gossip about Gossip"* e *"Virtual Voting"* para alcançar o consenso de forma eficiente e rápida. Isso elimina a necessidade de um consumo excessivo de energia e reduz o tempo necessário para confirmar as transações.

Embora a *Blockchain* enfrente desafios como ataques e limitações de escalabilidade, o *Hashgraph* tenta mitigar esses problemas através de seu design inovador. A estrutura *DAG* e os métodos de consenso ajudam a evitar o congestionamento da rede e a torná-la menos vulnerável a ataques. No entanto, como qualquer tecnologia emergente, o *Hashgraph* ainda está sujeito a desafios e críticas, e sua adoção e eficiência a longo prazo ainda estão sendo avaliadas.

2. Trabalhos Relacionados

Esta seção revisa alguns dos principais artigos que foram usados de base para o entendimento das tecnologias de *distributed ledger*, suas utilidades e seus desafios.

Inicialmente foi utilizado de base a *survey* [Hasanova et al. 2019] para um melhor entendimento dos conceitos que cercam a *blockchain* e os desafios para a adoção em larga escala da tecnologia, desde os problemas de segurança em cada versão quanto os problemas de escalabilidade.

A partir desse artigo, e o entendimento dos dilemas da escalabilidade e privacidade dentro das *blockchains*, foram pesquisadas alternativas para as implementações mais famosas da tecnologia como a *Bitcoin* e a *Ethereum*. Primeiramente, foram pesquisadas implementações alternativas do modelo DLT das *blockchains*, como no artigo [Salim et al. 2024] em que é proposto um modelo baseado em *Ethereum* mais escalavel e privado para o compartilhamento e processamento de dados médicos.

Depois, foram pesquisadas DLTs alternativas à *blockchain* e assim foi descoberta a tecnologia *Hashgraph* e suas particularidades. Uma vez decidido o enfoque do trabalho na tecnologia *Hashgraph* e a vontade de realizar uma implementação dela, foram usados como norte os documentos base para a criação do *Hashgraph* e da *Hedera* [Baird 2016], [Baird et al. 2019] que serviram para entender como se estabelecem as redes nessa tecnologia, como funciona a comunicação nelas e como é estabelecido o consenso para o funcionamento da DLT.

3. Visão geral do *Hashgraph* e Implementação

O *hashgraph* é uma estrutura de dados baseada em *DAG*. Diferentemente de uma *Blockchain*, que organiza transações em blocos lineares, o *hashgraph* registra transações (ou apenas a sincronização de informações usando o protocolo "*Gossip about Gossip*") como eventos que são representados como vértices em um grafo. Cada evento contém:

- **Carga útil (*payload*) de novas transações:** Detalhes sobre qualquer nova transação que o criador do evento desejou registrar naquele momento. Esses detalhes podem incluir informações como quem enviou a transação, quem a recebeu, o valor transferido, etc. É importante lembrar que a presença de uma carga útil é opcional. Nem todos os eventos precisam registrar transações alguns podem apenas registrar sincronizações de informações.
- **Carimbo da data e hora da criação do evento (*timestamp*):** Indica o momento exato em que o evento foi criado. Ele é crucial para ordenar os eventos temporalmente e para resolver conflitos de transação.
- ***Hash* do evento:** Cada evento contém um *hash* único, calculado a partir de seu conteúdo. Este *hash* é usado para determinar a posição relativa do evento no *hashgraph* e para verificar a integridade do evento.
- **Hashes de dois eventos anteriores:** Cada evento em um *hashgraph* referencia dois eventos anteriores, que são necessários para verificar o parentesco:
  - **Auto-pai (*Self-parent*):** um evento anterior criado pelo mesmo nó que está criando o novo evento.
  - **Outro-pai (*Other-parent*):** um evento anterior criado por um nó diferente, com o qual o criador do novo evento trocou informações.
- **Assinatura digital do criador do evento:** Garante a autenticidade do evento, identificando o nó que criou o evento na rede *hashgraph*.

3.1. *Gossip about Gossip*

O *Hashgraph* utiliza um protocolo de "Fofoca sobre Fofoca" (*Gossip about Gossip*) para a propagação de informações. Este protocolo é uma extensão do protocolo de "Fofoca" (*Gossip*) e funciona da seguinte maneira:

- **Fofoca (*Gossip*):** Quando um membro da rede, digamos "A", cria uma nova transação, ele seleciona aleatoriamente outro membro, digamos "B", e compartilha todas as informações que possui, incluindo a nova transação. Após esse compartilhamento inicial, tanto "A"quanto "B"irão, cada um, selecionar aleatoriamente novos membros da rede para compartilhar as informações recebidas. Esse processo de compartilhamento continua, com cada membro que recebe as informações escolhendo novos membros aleatoriamente para repassar a informação. Esse método de propagação, conhecido como "Fofoca"(*Gossip*), permite que as informações se espalhem exponencialmente pela rede. Em pouco tempo, todos os membros da rede terão conhecimento da nova transação criada pelo membro "A".

- **Fofoca sobre Fofoca (*Gossip about Gossip*):** Cada evento no *hashgraph* registra, além de seu próprio *hash*, os *hashes* de dois eventos anteriores: um do próprio membro, digamos ”A”e outro do membro que compartilhou informações com ele, digamos ”B”. Isso é essencial porque, além de compartilhar transações, os membros também compartilham informações sobre quem conversou com quem e em que ordem. Esse processo, conhecido como ”Fofoca sobre Fofoca”(*Gossip about Gossip*), cria um registro detalhado de todas as interações na rede. Esse registro é fundamental para a ”Votação Virtual”(*Virtual Voting*), que utiliza essas informações para alcançar o consenso de maneira rápida e eficiente. Com todos os membros possuindo uma visão completa e atualizada do *hashgraph* e utilizando a “Votação Virtual”, a rede pode determinar a ordem das transações de forma descentralizada, segura e precisa.

A figura abaixo demonstra como o *Hashgraph*, utilizando o protocolo de “Fofoca sobre Fofoca”, propaga informações:

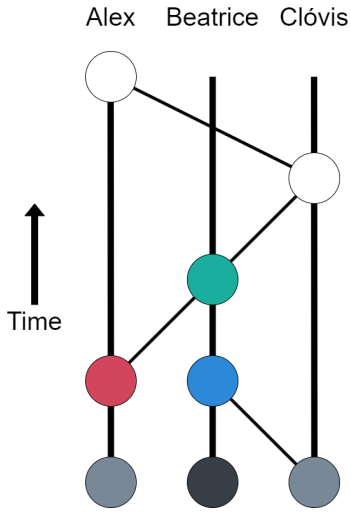

Figura 1. *hashgraph* simples com 3 usuários

O vértice verde no *hashgraph* representa um evento criado pela Beatrice. Nele, é registrado a ocorrência de Alex fazendo uma sincronização de informações com Beatrice (”Fofoca sobre Fofoca”), contando para ela tudo que ele sabe até o momento. Este evento também pode conter uma carga útil (*payload*) de novas transações além de conter o carimbo da data e hora da criação do evento (*timestamp*), a assinatura digital do criador do evento, o *hash* do evento, o *hash* do auto-pai (o evento representado pelo vértice azul) e o *hash* do outro-pai (o evento representado pelo vértice vermelho).

Os *hashes* dos eventos representados pelos vértices cinzas não são guardados no evento representado pelo vértice vermelho, mas são determinados por todos os outros *hashes*. O *hash* do evento representado pelo vértice cinza escuro é alcançável por sequências de auto-pais, enquanto dos eventos representados por vértices cinza-claro, não.

Ao final desta sincronização de informações, Beatrice irá possuir, no momento de criação deste evento, uma cópia completa do *hashgraph*. No entanto, não é suficiente que todos os membros conheçam todos os eventos. Também é necessário chegar a um

consenso sobre a ordenação linear dos eventos e, portanto, das transações registradas dentro deles. Iremos ver como realizar isso utilizando o *"Virtual Voting"*.

3.2. *Virtual Voting*

No contexto dos algoritmos tradicionais de consenso tolerantes a falhas bizantinas (*BFT*) sem líder, o consenso é frequentemente alcançado através de múltiplas rodadas de votação. Cada membro da rede envia seu voto para cada outro membro, resultando em um grande número de mensagens trocadas. Esse processo pode ser muito custoso em termos de largura de banda e tempo, especialmente à medida que o número de membros da rede cresce.

O *Hashgraph*, no entanto, evita esse *overhead* utilizando o *"Virtual Voting"*. Nesta abordagem, cada membro da rede mantém uma cópia do *hashgraph* completo (utilizando o protocolo de "Fofoca sobre Fofoca") . A partir desta estrutura de dados compartilhada, os membros podem calcular os votos que teriam sido enviados, sem realmente enviá-los.

A consistência entre as cópias do *hashgraph* de diferentes membros é garantida porque, eventualmente, todos os membros aprenderão sobre todos os eventos através da "Fofoca sobre Fofoca". No entanto, pode haver diferenças temporárias devido a eventos recentes que ainda não foram compartilhados com todos os membros. Essas pequenas discrepâncias são resolvidas à medida que o protocolo de "Fofoca sobre Fofoca" continua a funcionar.

Além de economizar largura de banda, o *"Virtual Voting"* também garante que os membros sempre calculem seus votos de acordo com as regras. Isso porque, caso um membro honesto "A" calcule os votos virtuais de um membro virtual honesto "B", mesmo que o verdadeiro membro "B" seja malicioso, ele não conseguirá interferir no cálculo dos votos virtuais de sua versão virtual que é honesta. No entanto, ele pode tentar realizar ataques de uma maneira diferente, através de *"Forks"*. Todavia, o *"Virtual Voting"* também garante proteção contra esses tipos de ataques através do conceito *"Strong Seeing"*.

3.3. Funcionamento do *Hashgraph*

3.3.1. Antecessor *(Ancestor)*

Dado um vértice "v" no *hashgraph*, qualquer vértice abaixo dele que estiver conectado com ele ou com um de seus vértices pais (auto-pai e outro-pai) será considerado um vértice antecessor. Podemos ver isso na Figura 1, onde tanto os vértices pais vermelho e azul quanto os vértices cinzas são vértices antecessores do vértice verde.

3.3.2. Consistente *(Consistent)*

Um membro "A" da rede pode ter uma cópia do *hashgraph* ligeiramente diferente da cópia do *hashgraph* de um membro "B" dado algum momento específico. Todavia, eles sempre serão "consistentes". "Consistente" significa que se ambos o *hashgraph* de "A" e o *hashgraph* de "B" contém o evento "x", então ambos irão conter o exato conjunto de antecessores de "x" e o exato conjunto de arestas entre esses antecessores.

3.3.3. Ver *(Seeing)* e *Fork*

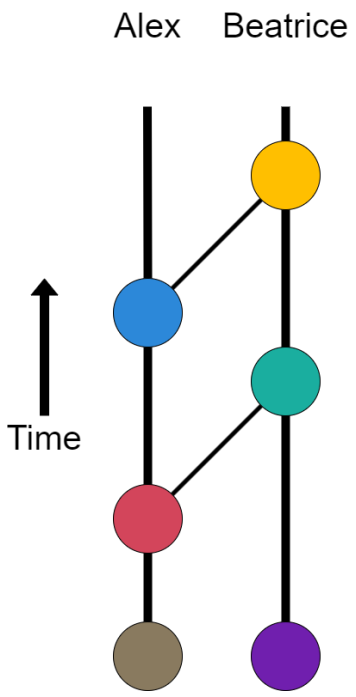

Figura 2. *hashgraph* simples com 2 usuários

Podemos dizer que o vértice verde "vê" o vértice roxo, vermelho e marrom, pois eles são antecessores dele. Todavia, o vértice azul não "vê" o vértice vermelho, pois ele é um *Fork* do vértice vermelho (ele foi criado a partir do vértice marrom, não sendo um evento que registra uma transação e/ou sincronização de informações com um outro membro da rede), em vez disso ele vê apenas o vértice marrom. Além disso, como o vértice amarelo tem como ancestrais os vértices azul e vermelho, e um é um *Fork* do outro, o vértice amarelo não vê nenhum dos dois, vendo apenas os vértices verde e roxo.

Com a utilização do Fork, um membro malicioso pode tentar manipular o resultado das votações no "Virtual Voting", por isso o *hashgraph* utiliza o conceito de "Ver Fortemente".

3.3.4. Ver Fortemente *(Strong Seeing)*, Testemunha *(Witness)* e Rodada *(Round)*

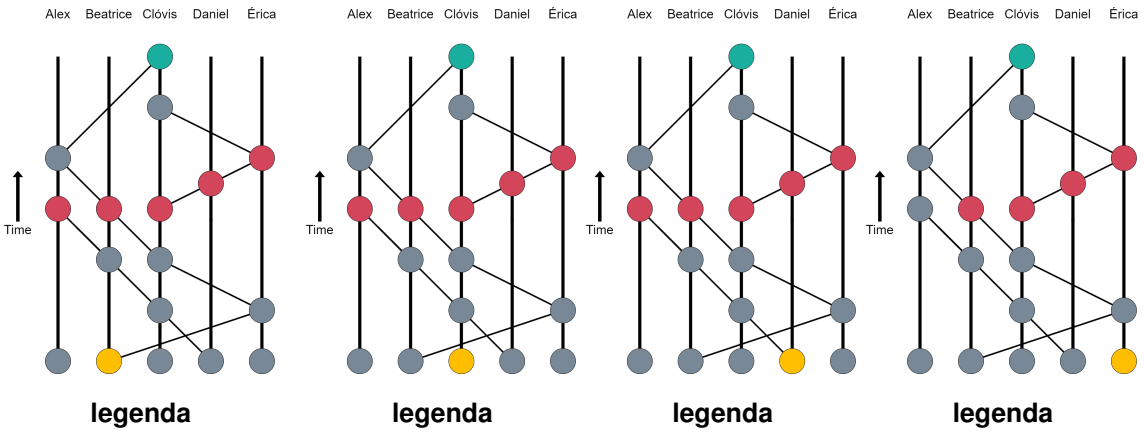

Figura 3. *hashgraphs* simples com 5 usuários

Em cada um dos *hashgraphs* acima, o vértice verde "Vê Fortemente" o vértice amarelo, pois ele "vê" mais que 2n/3 (sendo "n" a quantidade de membros na rede, 2*5/3 $\approx$ 4) vértices de membros diferentes, nesse caso os vermelhos, que por sua vez "vêem" o vértice amarelo.

Cada um dos vértices iniciais dos diferentes membros são chamados de "testemunhas" e a "rodada" destes vértices começa no número 0. Como o vértice verde "vê fortemente" mais que 2n/3, ou seja, 4 "testemunhas", ele é criado com "rodada" igual a 1. Além disso, como é o primeiro evento da "rodada" , o evento também é considerado a testemunha do membro para a "rodada" 1.

Só pode haver uma "testemunha" por membro em cada "rodada" (dado um membro específico, o primeiro de seus vértices que "vê fortemente" 4 "testemunhas" será uma "testemunha"). Para a rodada mudar novamente, o mesmo processo teria que se repetir.

4. Simulações e Testes

A partir do código implementado realizamos algumas simulações a fim de realizar algumas métricas. Dentre elas, trataremos especificamente de:

- **Taxa e taxa média de criação de eventos:** Eventos são criados a cada fofoca realizada, assim conseguimos medir, de certa forma, a vazão do *hashgraph*, verificando quantos eventos são criados por segundo. Com isso, conseguimos, também, analisar a escalabilidade da implementação proposta, aumentando ou diminuindo o número de usuários.
- **Média de eventos criados até que haja consenso:** Uma vez que um consenso é decido, possíveis transações são realizadas. Assim, com essa média conseguimos estipular uma taxa de transferência máxima para estas transações. Da mesma forma que no item anterior, conseguimos também analisar a questão de escalabilidade.

Destacamos que nossa implementação ela não é, de fato, paralela. Escolhemos lidar com esta simulação com as fofocas sendo feitas de forma sequencial por conta da

dificuldade e tempo extra para paralelizar a aplicação. Portanto, nossa análise será feita a partir de um *hashgraph* "piorado", o que não descaracteriza o propósito deste trabalho. Por fim, o código pode ser encontrado no repositório de um dos integrantes do grupo no link a seguir: Uma implementação do Hashgraph em Python.

4.1. Parâmetros

Para ambos testes, variamos o número de usuários N na rede. Detalhadamente, $N = 2$, $N = 4$, $N = 6$ e $N = 8$.

Além disso, em todas as simulações, precisamos focar as medições no *hashgraph* de um usuário específico. Isso é necessário, pois o grafo, apesar de convergir a um comum entre todos os usuários, a convergência não é instantânea. Assim, escolhemos, para todos os casos, o primeiro usuário da rede.

4.2. Taxa e taxa média de criação de eventos

Para a taxa de criação de eventos, calculamos, na verdade, o número de fofocas realizadas por segundo. Neste caso, medimos o intervalo de tempo Δt necessário para cem fofocas serem feitas, ou seja, obtemos o valor da taxa $T_e = \frac{100}{\Delta t}$. Além disso, a cada cem fofocas, definimos uma sequência de fofocas. Portanto, a primeira sequência engloba as cem primeiras fofocas, a segunda sequência, engloba as cem fofocas depois da primeira sequência, e assim por diante. Assim, na Figura 4 abaixo, observamos a taxa T_{e_i} para cada sequência $1 \leq i \leq 10$ de fofocas, para cada número de usuários N.

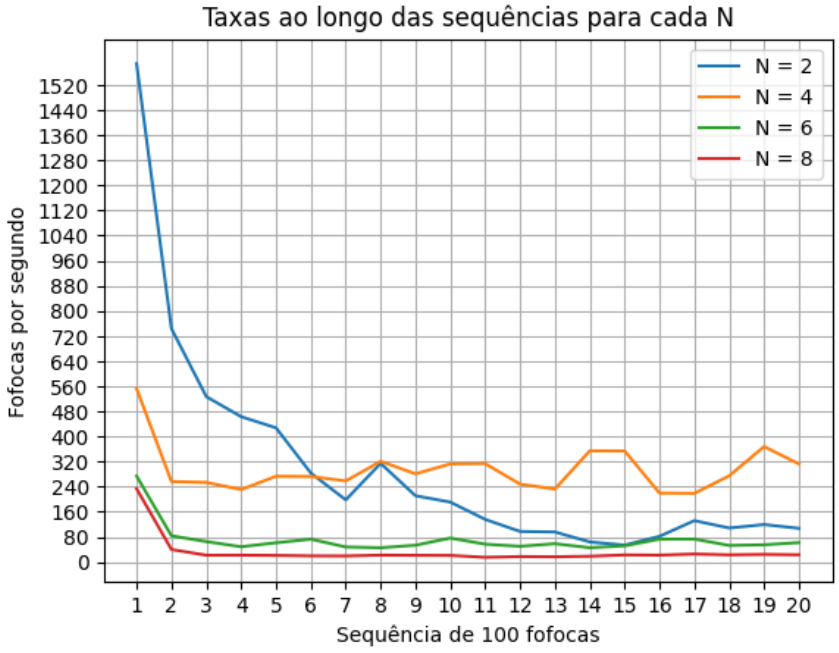

Figura 4. Taxas ao longo das sequências para cada N

Além disso, calculamos também a taxa média T_m da criação desses eventos, em dez sequências, a partir da expressão

$$T_m = \frac{1}{10} \sum_{i=1}^{10} T_{e_i}$$

Assim, a Figura 5 abaixo apresenta o gráfico com as taxas médias para cada número N de usuários.

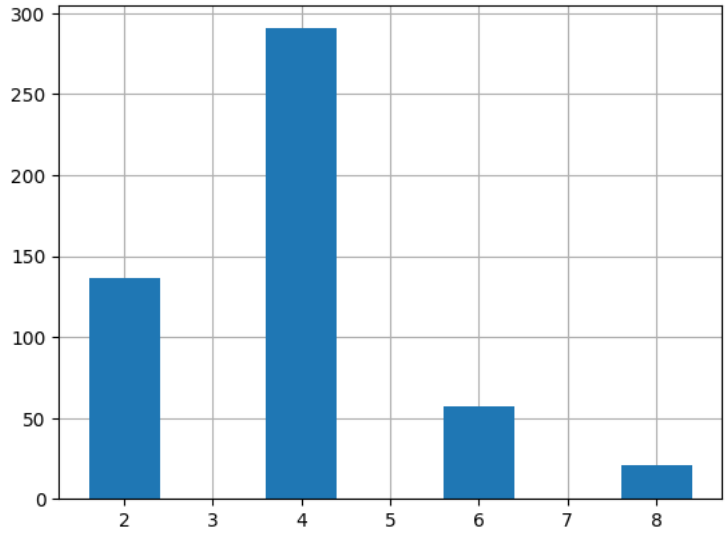

Figura 5. Taxas médias para cada N

4.3. Média de eventos criados até que haja consenso

Neste teste, contaremos os eventos criados até que um consenso é alcançado. Podemos definir como C_i o i-ésimo consenso e E_i o número de eventos criados entre C_{i-1} e C_i. Assim, a Figura 6 abaixo, apresenta o gráfico $C_i \times E_i$, para cada número N de usuários, com $1 \leq i \leq 10$.

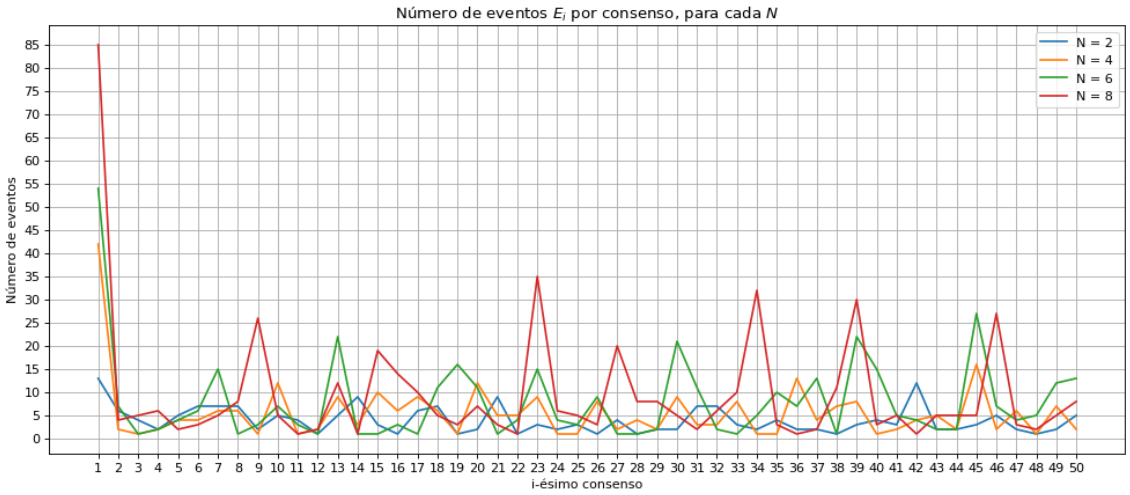

Figura 6. Número de eventos E_i por consenso, para cada N

Enfim, calculamos essa média de forma análoga ao item anterior. Ou seja,

$$M_e = \frac{1}{10} \sum_{i=1}^{10} E_i$$

Na Figura 7 podemos observar o gráfico com os valores das médias M_e para cada número N de usuários.

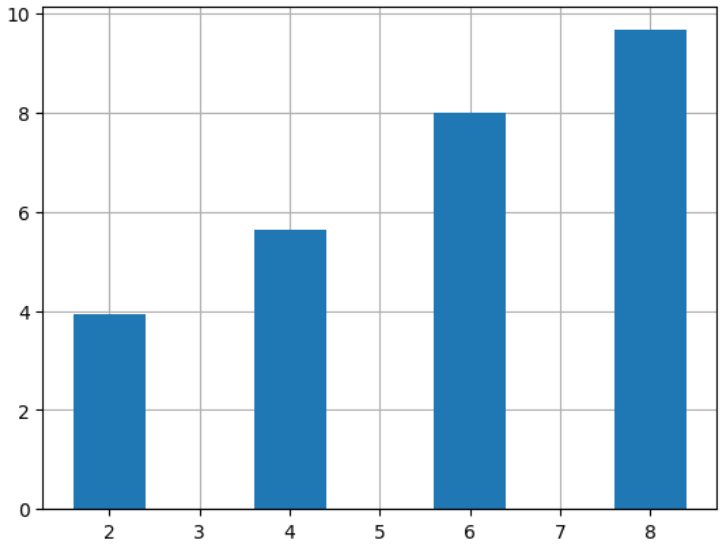

Figura 7. Média de eventos criados M_e, para cada N

5. Discussões

5.1. Resultados

Conforme aumentamos o número de usuários na rede, observamos que a taxa média de criação de eventos diminui e a média de eventos entre consensos aumenta. Essas duas variações são métricas que indicam piora no desempenho geral da rede. Assim, percebemos, que, para ambos os testes realizados, nossa implementação do *hashgraph* não apresentou a escalabilidade prevista no modelo original. No entanto, isso era o esperado, já que nossa simulação executa sequencialmente, em vez de paralelamente.

Além disso, na Figura 4 percebemos que as taxas se comportam como se fossem convergir para a média. Isso nos mostra que, com tempo suficiente, o desempenho da rede se mantém aproximadamente constante para cada número de usuários. Essa é uma observação relevante, pois indica que o tamanho do grafo, a partir de certo ponto, não afeta mais a taxa de criação de eventos. Da mesma forma, apesar de variar mais, a Figura 6, também indica uma convergência para a média de eventos entre consensos. Ou seja, nossa implementação, apesar de não escalar bem com o número de usuários, destaca o caráter escalável no tempo do *Hashgraph*: a taxa de conclusão de transações não piora com o aumento de eventos na rede.

5.2. Dificuldades

Durante a nossa pesquisa sobre o *Hashgraph*, notamos uma dificuldade entre artigos de detalhar seu funcionamento. Dentre todos, o que apresentou mais contribuição foi justamente a patente da tecnologia. No entanto, possuia uma estrutura muito pouco didática para o pouco que conhecíamos e para o tanto que precisávamos saber para implementar uma simulação. Assim, procuramos também por implementações prontas do *Hashgraph* em *Python* e encontramos, porém em C++, no repositório *Simple implementation of the consensus algorithm Hashgraph with clear visualizer*, que se baseou na patente.

Inicialmente a ideia era apenas se inspirar no código encontrado, para entender melhor como a tecnologia funciona, mas nem mesmo conseguíamos rodar a aplicação original. Por isso, vimos a necessidade de adaptar o código para *Python* e observá-lo executando. No entanto, não foi uma tarefa trivial. As duas linguagens têm suas peculiaridades que precisam ser consideradas durante a implementação. Por isso, muitos erros lógicos eram introduzidos no processo de adaptação. Tornando necessário, mais que justamente, reconsultar a patente para entender o que poderia estar dando errado. Então, ao longo da implementação muito, mas não tudo, foi entendido sobre o funcionamento do *Hashgraph*.

6. Conclusão

A partir de toda a pesquisa envolvida neste trabalho, assim como a implementação e resultados obtidos, percebemos mais uma vez a complexidade de algo que envolve *Blockchain*, mesmo com detalhes da implementação real. Apesar disso, dessa vez, com o *Hashgraph*, finalmente entendemos de forma mais concreta como funciona, justamente por conta da implementação.

Além disso, conseguimos obter dados condizentes com o esperado sobre a implementação sequencial do *Hashgraph*. O que nos permite pensar em trabalhos futuros, como sua paralelização e aplicação de outras melhorias que a patente também cita, como a *Fast elections* e *Efficient gossip*.

7. Referências

Baird, L. (2016). The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 34:9–11.

Baird, L., Harmon, M., and Madsen, P. (2019). Hedera: A public hashgraph network & governing council. *White Paper*, 1(1):9–10.

Hasanova, H., Baek, U.-j., Shin, M.-g., Cho, K., and Kim, M.-S. (2019). A survey on blockchain cybersecurity vulnerabilities and possible countermeasures. *International Journal of Network Management*, 29(2):e2060.

Salim, M. M., Yang, L. T., and Park, J. H. (2024). Privacy-preserving and scalable federated blockchain scheme for healthcare 4.0. *Computer Networks*, 247:110472.