Impacto do líder no desempenho do consenso BFT geo-distribuído*

Lucca Dornelles Cezar[1], Fernando Dotti[1], Fernando Pedone[2]

[1]PUCRS – Escola Politécnica [2]USI - Università della Svizzera italiana
lucca.dornelles99@edu.pucrs.br, fernando.dotti@pucrs.br, fernando.pedone@usi.ch

Abstract. *BFT consensus performance in blockchains is severely impacted by wide-area networks with a large number of participants. This paper analyzes the impact of leader selection on consensus efficiency and proposes a generic mechanism to prioritize proponents that optimize system throughput. The proposed solution is Byzantine-resilient and robust against network instability. Experimental results demonstrate that the mechanism achieves 80% of the performance of node-eviction techniques, without requiring node removal, thereby maintaining the protocol's original fault tolerance.*

Resumo. *O desempenho de blockchains com consenso BFT é severamente penalizado em redes geo-distribuídas com muitos participantes. Este artigo analisa o impacto da seleção de líderes na eficiência do consenso e propõe um mecanismo genérico para priorizar proponentes que otimizem a vazão do sistema. A solução é resiliente a falhas bizantinas e instabilidades de rede. Resultados demonstram que o mecanismo alcança 80% da eficiência de técnicas de remoção de nós, mas sem comprometer a tolerância a falhas original do protocolo.*

1. Introdução e Trabalhos Relacionados

Replicação Máquina de Estados (RME) [Lamport 1978, Schneider 1990] é uma abordagem importante para tolerância a falhas em sistemas distribuídos. Para sistemas compostos de processos não confiáveis, i.e. que podem se desviar arbitrariamente de sua especificação, RME tolerante a falhas bizantinas (BFT)[Castro et al. 1999] é empregada. Nesta classe encontram-se diversos tipos de sistemas, como de arquivos distribuídos [Adya et al. 2002], de bloqueio [Clement et al. 2009], para armazenamentos chave-valor [Bessani et al. 2013], e, mais recentemente, blockchains, tanto permissionadas como não permissionadas [Amiri et al. 2024]. Blockchains tendem a escalar em número de nós e em geo-distribuição, tanto para atender suas demandas quanto para favorecer sua disponibilidade e segurança. Por outro lado, o desempenho é impactado. Assim, testemunhamos nos últimos anos uma proliferação de propostas para escalar o consenso BFT e.g. [Xu et al. 2023, Hafid et al. 2020], desde diferentes perspectivas.

Grande parte dos algoritmos de consenso BFT utilizam um líder, que coordena o consenso de um conjunto de instâncias, podendo ser fixo, até a necessidade de troca por falha; ou rotativo, reduzindo a complexidade da troca de visão, buscando maior equidade na latência percebida por clientes, e aumentando a resistência à censura. Diversos trabalhos têm explorado o posicionamento do líder em sistemas geo-distribuídos

*Trabalho com suporte parcial do CNPq - processo 406993/2025-4.

[Huang et al. 2023, Wen and Yang 2025, Shan et al. 2025, Yu et al. 2024, Du et al. 2020]. Estes assumem que nós topologicamente distantes dos demais (em termos de latência) atrasam o consenso, não aprofundando este aspecto com dados de suporte. A maior parte destes propõe novos algoritmos, com melhorias sobre o PBFT, utilizando remoção de nós do comitê de consenso. Esses algoritmos buscam identificar nós bizantinos, ou apenas lentos, analisando dados de latência entre nós, obtidos de diferentes formas, ou comportamentos possivelmente bizantinos passados. Ao não separar questões de desempenho de comportamentos bizantinos, nós podem ser excluídos do comitê somente por baixo desempenho, prejudicando a tolerância a falhas. Ainda, existe uma lacuna de comparação com algoritmos modernos, que empregam detecção de bizantinos e rotação de líder.

Neste artigo, discutimos, com base em resultados experimentais, os impactos do posicionamento do líder em cenários geo-distribuídos. A partir disso, evitando mais uma proposta de algoritmo de consenso, apresentamos um mecanismo genérico para priorizar líderes rápidos. Discutimos que este mecanismo é tolerante à ação de nós bizantinos e apresentamos os resultados de desempenho obtidos, mostrando os ganhos quando aplicado a um sistema moderno com rotação de líder e mecanismos de detecção e remoção de nós bizantinos.

2. Algoritmo Base e Modelo de Sistema

Adotamos o Tendermint [Buchman 2016] e seu respectivo modelo de sistema como representativo de uma classe de algoritmos de consenso para blockchains. Consideramos um conjunto Π com N processos que se comunicam por mensagens, não compartilham memória e não têm acesso a um relógio global. Até $f < N/3$ processos podem se desviar de sua especificação, sendo chamado de faltosos ou bizantinos, os demais são ditos honestos ou corretos. Assume-se o mesmo modelo de sincronia parcial e a utilização de técnicas criptográficas, sendo que adversários não terão (com alta probabilidade) poder computacional para subverter as mesmas.

Tendermint é baseado no PBFT [Castro et al. 1999], introduzindo um comitê de decisão, montado a partir de prova de participação (*proof of stake*), e liderança rotativa, ou seja, cada instância (ou altura) de consenso é proposta por um nó do comitê, em uma sequência determinada por participação. O consenso baseia-se em fases cíclicas: o proponente da altura difunde a proposta ($propose$) do bloco; ao recebê-la, cada nó difunde seu voto inicial ($prevote$) a todos; ao receber mais de $2/3$ de prevotes, o nó bloqueia o valor e difunde seu voto de confirmação ($precommit$); finalmente, ao receber mais de $2/3$ de precommits, o nó atinge a finalidade (commit), efetivando o bloco e preparando-se para a próxima altura com um novo proponente.

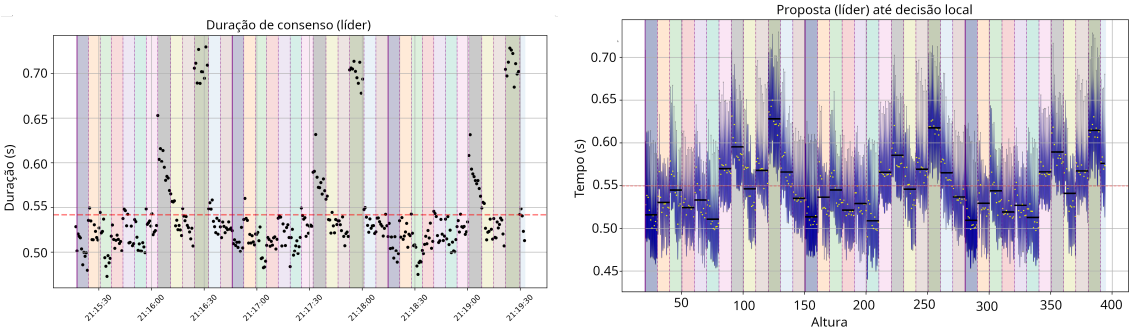
3. Estudo do impacto do líder

$Metodologia.$ Utilizamos o ambiente CloudLab[1] com nós sincronizados via NTP[2] e uma implementação Tendermint[3] adaptada com a capacidade de definir a sequencia de proponentes. Todos experimentos duram 300 segundos, descontam-se os 30 primeiros de aquecimento e os 15 últimos, com pelo menos duas execuções por configuração e comparação

[1]https://www.cloudlab.us/
[2]https://www.ntp.org/
[3]Malachite Starknet V0.4.x https://github.com/circlefin/malachite/tree/v0.4.x

de resultados. Experimentos configurados com nós computacionais de mesma capacidade e arestas de comunicação com latências iguais mostram que o desempenho do sistema é o mesmo para proponentes fixos ou rotativos, independente do seu posicionamento. Nos demais experimentos, para cenários geo-distribuídos, utilizamos as latências entre nós conforme 13 zonas da AWS[4] com um nó em cada zona.

Achados. A Figura 1(a) apresenta amostras de latência de consenso. Para maior facilidade de visualização, cada proponente realiza 10 propostas (o desempenho é equivalente ao funcionamento padrão de troca a cada proposta), sendo suas respectivas amostras separadas por linhas roxas pontilhadas verticais. As linhas roxas sólidas indicam retorno ao proponente inicial, repetindo a sequencia de proponentes. Certos proponentes consistentemente levam a consensos mais rápidos (e.g. 1º e 3º proponentes) ou mais lentos (e.g. 8º e 11º). Considerando que, em relação aos experimentos anteriores, somente a matriz de latências foi modificada, atribuem-se os diferentes tempos de consenso exclusivamente ao posicionamento dos respectivos proponentes. Complementando, na Figura 1(b) cada linha azul é um conjunto de pontos verticais referentes ao tempo entre o início do consenso no proponente até a decisão local de cada nó da rede, bem como sua média por consenso (amarelo) e por proponente (preto). Novamente podemos observar os mesmos padrões, mostrando que a posição do proponente impacta a performance de todos os nós da rede.



(a) Latência do consenso em cada proponente.

(b) Variação de tempo de decisão em cada participante.

Figura 1. Tempos para decisão no proponente e nos participantes.

Experimentos adicionais mostram que o proponente impacta também os tempos das fases de prevote e precommit em cada nó. Isto decorre do algoritmo já que cada nó deve reunir um quorum de mensagens da fase, e a chegada destas depende do momento em que cada nó recebeu o propose do proponente, para então reagir. Adicionalmente, avaliamos a frequência de cada nó nos quorums formados em cada instância de consenso[5]. Avaliamos dois períodos em que cada nó atuou como proponente por 10 rodadas consecutivas. Entre esses dois períodos, identificamos alta variabilidade na composição do quorum entre diferentes proponentes, com grande sobreposição (98%) na participação de nós em quorums ao comparar os dois períodos de mesmo proponente. Esse resultado sugerem regularidade na formação de quorums e dependência no proponente.

[4]https://github.com/cometbft/cometbft/blob/v2.x/test/e2e/pkg/files/aws-latencies.csv
[5]Quorums são relativos e nós podem ter quoruns diferentes no mesmo consenso.

4. Priorização de proponentes rápidos

Propomos um mecanismo que altera a frequência com que cada nó atua como proponente, favorecendo nós que melhoram a vazão do consenso. O mecanismo é intencionalmente ortogonal ao consenso, resiliente a nós bizantinos e não modifica as suposições do consenso ou do comitê, facilitando sua integração a diferentes algoritmos.

Nosso mecanismo é baseado em consenso sobre observações locais. Cada nó monitora a duração de cada consenso, podendo então classificar cada instância em um valor $v \in \{lento, \perp, rápido\}$ a partir da sua média local e uma variável T. Um consenso é considerado lento ou rápido quando a sua duração desvia da média por mais de $T\%$. Para que os nós gerem uma observação global a partir das locais, utilizamos um consenso de três fases com quorum bizantino onde cada etapa dura uma instância do consenso em uso (e.g. Tendermint neste caso), sendo as mensagens enviadas juntamente da primeira mensagem de cada consenso. As etapas são: ❶ votação, onde nós disseminam seus votos v locais; ❷ decisão, onde nós difundem o valor v aceito por um quorum, ou $\perp$ se não formarem um quorum; ❸ escrita onde o proponente escreve o valor v, aceito por um quorum de decisões, no bloco, ou $\perp$ caso não receba um quorum. Para prevenir líderes bizantinos de sobrescreverem decisões ao registrar no bloco, os nós só aceitam o bloco proposto em ❸ com valor v se o quorum recebido em ❷ concordar com este valor. Com isso, a classificação atualizada de cada nó é registrada para todos participantes que a utilizam deterministicamente para definir a sequencia de proponentes. Nosso algoritmo ainda define uma parcela da participação (*stake*) associada a desempenho, sendo este valor configurável. Quando essa parcela é inferior a 100%, todos os nós eventualmente serão proponentes, mitigando problemas de censura.

Esta proposta de classificação de nós deve ser tolerante a bizantinos. Observamos que, em cada fase, a classificação de um nó em *rápido* ou *lento* só ocorre se $2f + 1$ nós concordam com este valor. Caso contrário a classificação fica $\perp$, que equivale a não alterar a prioridade do nó. Analisando a relação entre nós honestos e desonestos, observamos que: se $2f + 1$ ou mais honestos votam em um valor, ele será aceito; se menos de $f + 1$ honestos votam em um valor, ele não será aceito; e quando temos entre $f + 1$ e $2f + 1$ votos de honestos em um valor, este pode ser aceito dependendo do posicionamento dos demais f, possivelmente desonestos. Caso os votos de desonestos ajudem a completar $2f + 1$ votos por um valor, este valor será decidido, o que consideramos aceitável pois este valor foi votado pela maioria de honestos. Caso contrário o valor não será aceito e $\perp$ é decidido, impedindo a diferenciação do nó em questão dos demais, o que mantém o funcionamento do algoritmo subjascente para o respectivo nó. Ou seja, os nós desonestos não tem a possibilidade de prejudicar o desempenho do sistema utilizando esta proposta.

Validamos o nosso algoritmo utilizando a mesma metodologia da Seção 3, com 5 execuções por configuração. Na Figura 2, na coluna def temos todos nós com peso igual e rotatividade round-robin para a escolha de proponente. Nas colunas de alg_1 a alg_6, cada linha é uma série de execuções obtida como segue. Executamos o sistema por 300 segundos, após a execução simulamos o voto local de cada nó (conforme mecanismo descrito acima) e calculamos a decisão global ($2f + 1$ votos pelo mesmo valor, ou $\perp$) a respeito de cada nó. Esta configuração é empregada no cálculo de participações para a próxima configuração (alg_1). Este procedimento continua indutivamente para i de 1 a 5, tomando-se "alg_i"como inicial e obtendo-se a configuração para alg_{i+1}. Para tornar o

sistema resistente a flutuações na rede não usamos apenas a última decisão, mas sim o histórico de decisões recentes para calcular a sequencia de proponentes. A média de cada configuração é dada pela linha vermelha.

Em $rem2$ mostra-se o equivalente a uma técnica de remoção de nós do comitê, obtida ao remover dois nós desproporcionalmente lentos dessa topologia (observáveis na Figura 1(a)). Nesse experimento podemos ver que ao melhorar o funcionamento, o algoritmo altera a média até o ponto onde se estabiliza. Nesse ponto nosso algoritmo tem o ganho equivalente a 80% da estratégia que remove os nós lentos do comitê. Entretanto, nossa técnica não afeta o comitê; enquanto a técnica de remoção implica em perda de 25% da tolerância bizantina nesse experimento, pois são removidos 2 nós de 13, baixando a tolerância de 4 para 3 bizantinos.

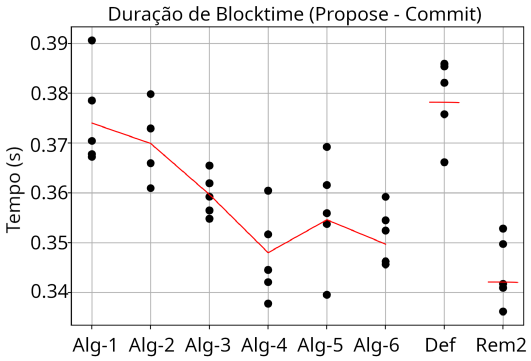


Figura 2. Experimentos com o mecanismo de seleção de proponente.

Também utilizamos dados de [Di Perna et al. 2025] para gerar uma segunda topologia com variabilidade nas latências. Neste caso observamos um ganho de 16% relativo ao sistema rotativo. Fazendo-se uso do histórico de decisões, notamos maior resiliência a variabilidades na rede. Para complementar a avaliação também executamos experimentos em topologias com variabilidade extrema ($>$100ms) nas latências onde o algoritmo resultou em ganhos menores, sem comprometer o desempenho do sistema.

5. Conclusão

Este trabalho demonstrou que a seleção estratégica de líderes em redes geo-distribuídas otimiza o consenso BFT sem comprometer a segurança do protocolo. As principais contribuições incluem a caracterização experimental do impacto da latência sobre proponentes específicos e a proposição de um mecanismo de priorização ortogonal e resiliente a falhas bizantinas. A solução alcançou 80% da eficiência de métodos que removem nós, porém mantendo a tolerância a falhas inalterada. Em redes com alta variabilidade de atraso, observou-se um ganho de 16%, enquanto em cenários onde a diferenciação de nós não é possível, a técnica preserva o desempenho do sistema original.

Referências

Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P. (2002). Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14.

Amiri, M. J., Wu, C., Agrawal, D., Abbadi, A. E., Loo, B. T., and Sadoghi, M. (2024). The bedrock of byzantine fault tolerance: A unified platform for BFT protocols analysis, implementation, and experimentation. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 371–400, Santa Clara, CA.

Bessani, A., Correia, M., Quaresma, B., André, F., and Sousa, P. (2013). Depsky: dependable and secure storage in a cloud-of-clouds. *Acm transactions on storage (tos)*, 9(4):1–33.

Buchman, E. (2016). Tendermint: Byzantine fault tolerance in the age of blockchains. Master's thesis, University of Guelph.

Castro, M., Liskov, B., et al. (1999). Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186.

Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M., et al. (2009). Making byzantine fault tolerant systems tolerate byzantine faults. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 153–168. The USENIX Association.

Di Perna, V. P., Bernardo, M., Fabris, F., Amaro, S., Matos, M., and Schiavoni, V. (2025). Impact of network topologies on blockchain performance. In *Proceedings of the 19th ACM Int. Conf. on Distributed and Event-based Systems*, pages 122–133.

Du, N., Liang, Z., Huang, Y., Guo, Z., Yang, H., and Wang, S. (2020). Performance optimisation method of pbft consensus for supply chain integration svm. In *2020 7th international conference on dependable systems and their applications (DSA)*, pages 371–377. IEEE.

Hafid, A., Hafid, A. S., and Samih, M. (2020). Scaling blockchains: A comprehensive survey. *IEEE Access*, 8:125244–125262.

Huang, D., Huang, Y., and Yang, Y. (2023). Improved pbft consensus algorithm based on node evaluation and dynamic management. In *Proceedings of the 2023 4th International Conference on Big Data Economy and Information Management*, pages 851–855.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

Shan, B., Gao, T., Song, L., and Cui, Y. (2025). Grouped byzantine fault-tolerant consensus mechanism based on node behaviour analysis. In *2025 4th International Symposium on Computer Applications and Information Technology (ISCAIT)*, pages 1890–1895. IEEE.

Wen, X. and Yang, X. (2025). Mapbft: multilevel adaptive pbft algorithm based on discourse and reputation models. *The Computer Journal*, 68(6):635–648.

Xu, J., Wang, C., and Jia, X. (2023). A survey of blockchain consensus protocols. *ACM Comput. Surv.*, 55(13s).

Yu, L., Wu, Y., Lu, J., and Li, T. (2024). An adaptive reputation update mechanism for primary nodes in pbft. In *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1948–1953. IEEE.