Análise comparativa do algoritmo Paxos e suas variações

Sofia Regis[1], **Odorico M. Mendizabal**[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
Campus Universitário Reitor João David Ferreira Lima – Florianópolis – SC – Brasil

sofia.regis@grad.ufsc.br, odorico.mendizabal@ufsc.br

Abstract. *Distributed consensus protocols are fundamental building blocks for developing dependable distributed systems. They help design systems with high availability and strong consistency guarantee, even in the presence of failures in some participants. Paxos is the main consensus algorithm of the last decades, and several variations and optimizations have been presented since the original algorithm was proposed. This paper surveys the main variations and optimizations of the Paxos algorithm, emphasizing the differences, advantages, and disadvantages between the variants and the traditional protocol.*

Resumo. *Protocolos de consenso distribuído são blocos de construção fundamentais para o desenvolvimento de sistemas distribuídos confiáveis. Eles auxiliam na elaboração de sistemas com alta disponibilidade e garantia de consistência forte, mesmo no caso de falhas em alguns participantes. O Paxos é o principal algoritmo de consenso das últimas décadas com diversas variações e otimizações sendo pesquisadas e desenvolvidas desde que o algoritmo original foi proposto. Este artigo apresenta um levantamento sobre as principais variações e otimizações do algoritmo Paxos, observando diferenças, vantagens e desvantagens entre as variantes e o protocolo tradicional.*

1. Introdução

Protocolos de consenso distribuído são utilizados para assegurar que múltiplos processos cheguem a um acordo sobre a escolha de um valor proposto. Eles são empregados com o intuito de prover alta disponibilidade e assegurar consistência, permitindo que uma coleção de processos distribuídos trabalhem como um grupo coerente mesmo no caso de falhas de alguns participantes. Os protocolos de consenso funcionam como blocos de construção para o desenvolvimento de diversos serviços, como eleição de líder, exclusão mútua, difusão atômica, replicação ativa, entre outros.

Pesquisas sobre consenso distribuído são maduras, com importantes algoritmos propostos nos anos 90, como o Paxos [Lamport 1998] e *Viewstamped Replication* [Oki and Liskov 1988], até alternativas mais recentes, como o Zab [Junqueira et al. 2011] e o Raft [Ongaro and Ousterhout 2014]. O algoritmo Raft, por exemplo, é implementado em aplicações bastante conhecidas, como o etcd, utilizado internamente pelo orquestrador de contêineres Kubernetes, e o Kudu, motor de armazenamento de dados usado pelo Hadoop. O Hadoop utiliza o serviço Zookeeper, construído sobre o protocolo de consenso Zab, e é usado para armazenar configurações consistentes entre réplicas e permitir recuperação de falhas automática. Outros sistemas desenvolvidos com o apoio de protocolos de consenso são o Chubby, utilizado pelo *Google File System* e BigTable.

Paxos pode ser considerado o principal algoritmo de consenso distribuído nas últimas décadas. Apesar da sua popularidade, este não é um algoritmo simples de compreender. Foram publicadas diversas extensões e trabalhos complementares com o intuito de explicá-lo e facilitar a sua implementação em sistemas práticos [Lamport 2001, Chandra et al. 2007]. A versão original do Paxos implementa um algoritmo de consenso de duas fases e os processos podem exercer três papéis distintos, que são: *proposers*, *acceptors* e *learners*[1]. Os *proposers* podem sugerir valores e os *acceptors* são responsáveis pela escolha de um único valor para uma rodada de consenso. Dessa forma, será garantido que os *learners* receberão os mesmos valores em cada rodada de consenso.

Nos últimos anos, otimizações e variações foram criadas baseadas no Paxos, visando, por exemplo, redução no número de rodadas de mensagens trocadas para início de uma escolha de valor ou redução nos custos com replicação dos processos participantes [Lamport and Massa 2004, Lamport 2006]. Algumas variantes relaxam a ordem de entrega de comandos sucessivos, sem violar critérios de consistência [Pedone and Schiper 1999, Lamport 2005]. Mais recentemente, o CASPaxos [Rystsov 2018] propõe uma versão do protocolo sem a replicação de histórico de operações.

Implementações na forma de bibliotecas e ferramentas que implementam consenso baseado em Paxos foram desenvolvidas e estão disponíveis. Algumas aplicam otimizações baseando-se nas variações do algoritmo Paxos ou em escolhas de tecnologias (p. ex. comutador de rede programáveis [Dang et al. 2020]) ou plataformas de propósito específico independentes do algoritmo (p. ex. uso de múltiplas interfaces de rede [Marandi et al. 2012]). Alguns exemplos de bibliotecas que implementam Paxos, prontas para uso são: Multi-Ring Paxos[2], BFTSMaRt[3], LibPaxos[4], módulo Paxos no Cassandra[5] e phxPaxos[6].

Apesar do grande avanço nas pesquisas sobre o protocolo Paxos e no desenvolvimento de bibliotecas práticas deste protocolo, requisitos atuais para sistemas distribuídos ainda demandam otimizações e adaptações neste protocolo. Por exemplo, serviços em plataformas de computação em nuvem compartilham recursos e frequentemente são submetidos a estratégias de relocação e migração, necessitando agilidade na recuperação e reconfiguração do grupo de participantes; Aplicações de IoT e *Blockchains* podem ter um grande número de participantes, demandando soluções de alta escalabilidade; serviços de processamento de lotes ou fluxo de dados estão sujeitos à uma grande taxa de requisições, logo protocolos de consenso subjacentes devem atender alta vazão de requisições. Dessa forma, a compreensão sobre o protocolo e os seus potenciais para adaptações, limites e otimizações são cruciais para o avanço nesta área de pesquisa.

O objetivo deste artigo é fazer um levantamento sobre diferentes variações do algoritmo Paxos e suas implementações. Este trabalho tem o intuito de descrever as

[1]Adotamos a nomenclatura original para designar os papéis no protocolo Paxos. Uma tradução livre para *proposers*, *acceptors* e *learners* pode ser dada por: proponentes, aceitadores e aprendizes.

[2]Multi-Ring Paxos: `https://github.com/sambenz/URingPaxos`

[3]BFTSMaRt: `https://github.com/bft-smart/library`

[4]Libpaxos: `https://bitbucket.org/sciascid/libpaxos/src/master/`

[5]Cassandra Paxos: `https://github.com/apache/cassandra/tree/trunk/src/java/org/apache/cassandra/service/paxos`

[6]phxPaxos: `https://github.com/Tencent/phxpaxos`

otimizações e mudanças propostas nos últimos anos, fazendo uma comparação observando quais aspectos do protocolo variam em relação a estrutura tradicional do protocolo Paxos. Apesar de existirem trabalhos comparativos sobre protocolos de consenso, eles limitam-se a poucas variantes e, na sua maioria, as comparações relacionam Paxos com outros protocolos (e.g. Raft, Zab e *Viewstamped Replication*). Além de apresentar um estudo detalhado sobre o protocolo Paxos e suas variações, até onde os autores puderam identificar, este é o primeiro artigo comparativo sobre o tema em língua portuguesa. Assim, este manuscrito também contribui como base para iniciação e direcionamento de pesquisas envolvendo Paxos em países de língua portuguesa. Esta pesquisa utilizou repositórios de artigos acadêmicos. Devido a limitações de espaço, variantes bizantinas do Paxos não foram incluídas.

O restante do artigo está organizado como segue. A Seção 2 introduz o protocolo Paxos e suas principais variações. A Seção 3 descreve a estrutura base do algoritmo Paxos e destaca as alterações nas diferentes variações. Finalmente, a Seção 4 conclui o trabalho.

2. Paxos e suas variações

O Protocolo de consenso Paxos [Lamport 1998, Lamport 2001] foi proposto por Leslie Lamport em 1998 e consiste em decidir um único valor entre um conjunto de valores propostos por processos em um sistema assíncrono, onde processos podem falhar e mensagens podem ser perdidas. Existem 3 papéis possíveis para processos executando o protocolo. Cada processo pode desempenhar mais de um papel ao mesmo tempo:

- ***Proposer*:** Processo que deseja ter um determinado valor escolhido;

- ***Acceptor*:** Processo que concorda com a aceitação de valores propostos. Este processo persiste valores aceitos para permitir a recuperação e identificação de um valor. A decisão de um valor é feita com base no valor aceito por um quórum de *acceptors*;

- ***Learner*:** Processo que deseja aprender o valor decidido.

A qualquer momento, um *proposer* pode propor um valor. A proposta de um valor ocorre em duas fases: *fase 1* e *fase 2*. Cada fase requer a confirmação de uma maioria dos *acceptors* para conclusão.

Fase 1 – *Prepare* e *Promise*: Um *proposer* escolhe um número de proposta único b (b é usado para representar um *ballot* conforme a nomenclatura original do protocolo) e envia uma mensagem $prepare(b)$ para os *acceptors*. Cada *acceptor* sem falha recebe $prepare(b)$ e se b for o maior número de proposta já recebido, então b é escrito em armazenamento persistente e o *acceptor* responde enviando a mensagem $promise(b, \perp)$, onde b indica que o *acceptor* se compromete a aceitar um valor para a proposta b e $\perp$ significa que nenhum valor foi aceito ainda. Caso o *acceptor* já tenha aceito um valor para alguma proposta b' menor ou igual a b, ele responde enviando a mensagem $promise(b', v')$, sendo b' o número da proposta e v' o valor aceito naquela proposta. No caso da proposta b ser menor que b', o *acceptor* simplesmente ignora a requisição sem responder ao *proposer*.

Se o *proposer* receber mensagens $promise$ de um quórum contendo a maioria dos *acceptors*, ele procede para *fase 2*. Caso não haja uma maioria de respostas, em um dado tempo, ele pode propor o valor novamente com um novo número de proposta b'', maior que b. Ao fazer uma nova proposta, é possível que *acceptors* respondam se comprometendo a aceitar um valor para a nova proposta ou informando o valor que se comprometeram em

alguma proposta anterior, por meio de uma mensagem $promise(b', v')$. Neste último caso, o *proposer* entende que há uma proposta em curso, e atualizará o valor v a ser proposto com o valor v' associado à proposta b' de maior valor informada pelos *acceptors*.

Fase 2 - *Accept* e *Accepted*: O *proposer* deve informar o valor v para os *acceptors*. Se mais de um valor foi retornado pelos *acceptors* na *fase 1*, então o *proposer* escolhe o valor v' associado ao maior número de proposta. Se nenhuma proposta em curso anteriormente for retornada, então o *proposer* enviará o valor v, o qual tinha a intenção de propor inicialmente. O *proposer* envia a mensagem $accept(b, v)$ para os *acceptors*.

Cada *acceptor* sem falha recebe um $accept(b, v)$ e se b for igual ou maior ao seu número de proposta armazenado, então a proposta recebida é escrita em armazenamento persistente e o *acceptor* responde com $accepted(b, v)$. Assim que o *proposer* receber $accepted(b, v)$ da maioria dos *acceptors*, ele conclui que o valor v foi decidido. A disseminação do valor decidido para os *learners* pode ser feita (i) pelo *proposer* ao concluir que o valor foi decidido ou (ii) pelos próprios *acceptors* que, além de enviar a mensagem $accepted(b, v)$ para o *proposer*, a enviariam também para os *learners*.

Uma implementações do protocolo Paxos deve satisfazer as seguintes propriedades de segurança no funcionamento (*safety*):

- *Não-trivialidade:* apenas um valor que tenha sido proposto pode ser aprendido;
- *Estabilidade:* um *learner* aprende no máximo um valor (em outras palavras, ele não muda de ideia sobre um valor aprendido);
- *Consistência:* Dois *learners* não podem aprender valores diferentes.

Com isso, o Paxos garante que quando um valor foi decidido, a decisão é final e um valor diferente não pode ser escolhido. Paxos chega a uma decisão desde que $((n/2) + 1)$ entre os n *acceptors* estejam funcionando corretamente. O Paxos não garante progresso (*liveness*) em um sistema completamente assíncrono[7], então é necessário assumir que o sistema eventualmente irá se comportar de forma síncrona por um período suficiente para que o protocolo chegue a uma decisão.

2.1. Multi Paxos

Uma maneira de otimizar o protocolo para permitir a escolha de uma sequência de valores em diferentes instâncias é chamada de Multi Paxos. O principal objetivo dessa otimização é permitir que não apenas um valor v seja escolhido, mas que uma sequência de valores $v_1 \ldots v_n$ possa ser determinada em consenso. Além disso, esta otimização visa reduzir o número de fases do protocolo e, consequentemente, de mensagens trocadas.

A *fase 1* do Paxos é independente do valor que será proposto e portanto pode ser executada antes do *proposer* saber qual valor ele irá propor. Então, o *proposer* pode enviar uma mensagem *prepare* que faz uma "reserva" de n instâncias Paxos, e se uma maioria de *acceptors* concordar, esse *proposer* não irá mais executar a *fase 1* enquanto a reserva não se esgotar. Com isso, ele poderá executar diretamente a *fase 2* nas n instâncias seguintes.

A reserva de n instâncias feitas por um *proposer* p_1 pressupõe que este *proposer* seguirá realizando até n envios de valores na *fase 2*. Ocorre que um *proposer* p_2 pode ter

[7]O famoso resultado de Fischer, Lynch e Paterson (FLP) [Fischer et al. 1985] demonstra que nenhum algoritmo distribuído em sistema assíncrono pode resolver o problema de consenso na presença de falhas.

um comportamento parecido, enviando mensagens de reserva de instâncias com valores mais altos, o que faria com que os *acceptors* passassem a ignorar as mensagens de *fase 2* do *proposer* p_1, visto que estes estariam comprometidos com instâncias de p_2. De forma análoga, outro *proposer* p_3 ou mesmo p_1 poderia reservar novas instâncias de maior valor que as reservadas por p_2, prejudicando o progresso das propostas de p_2. Esse efeito é conhecido como *colisões*, postergando a decisão de valores e causando a alternância entre *proposers* que estejam propondo valores.

Para reduzir as chances de colisões, esta variação do protocolo Paxos admite o papel de *coordenador* ou *líder*[8]. O *coordenador* é um dos *proposers* em execução e apenas este *proposer* pode propor valores diretamente aos *acceptors*. Dessa forma, enquanto o coordenador estiver propondo valores, os demais *proposers* (i) não propõe valores ou (ii) encaminham os seus valores ao coordenador, para que este faça as propostas aos *acceptors*. O papel de *coordenador* pode ser alternado entre os *proposers* e, no caso falha do *coordenador*, outro *proposer* deve ser eleito *coordenador*.

2.2. Fast Paxos

O Fast Paxos [Lamport 2006] é uma extensão do Paxos tradicional que procura reduzir os número de passos necessários para chegar no consenso. Para tal, os tipos de instâncias de consenso são expandidas e divididas na versões "clássica" e "rápida" que são diferenciadas de acordo com o número de proposta, permitindo ao *proposer* líder coordenar o tipo de instância que será executada.

A versão clássica se comporta como o Paxos tradicional, porém a rápida tem algumas diferenças, e a principal é que o *proposer* principal pode enviar uma mensagem do tipo *any* para os *acceptors* na *fase 2a*, denotando que eles podem aceitar valores propostos de quaisquer *proposers* mesmo que não sejam do líder ou que já existam *acceptors* aceitando outros valores. Ao permitir que *acceptors* se comprometam com diferentes valores ao invés de precisarem respeitarem sempre os mesmos valores é introduzido o risco de colisão, onde o valor pode não ser decidido. A recuperação de colisões pode ser feita ao repetir a proposta utilizando a versão de consenso clássica.

2.3. Dynamic Paxos

O Dynamic Paxos [Lamport and Massa 2004] possibilita que os conjuntos de *proposers*, *acceptors* e *learners*, assim como os participantes do quórum de *acceptors*, possam ser modificados em tempo de execução. Isso permite que o sistema se reconfigure, dando flexibilidade para o algoritmo mudar a composição dos participantes do algoritmo.

A informação atual sobre os conjuntos precisa ser armazenada pela aplicação que utiliza o protocolo. Por exemplo, uma implementação de Replicação por Máquina de Estados (RME) [Schneider 1990] usando protocolo de consenso é comum, sendo que nesta abordagem cada réplica pode executar os papéis de *proposer*, *acceptor* e *learner*. Dessa forma, as informações sobre os os conjuntos modificadas durante a execução do algoritmo de consenso podem ser salvas no estado das máquinas de estado replicadas.

[8]A nomenclatura para distinguir o *proposer* com papel diferenciado varia na literatura, sendo comum encontrar os termos *coordenador* ou *líder*. Neste artigo ambos os termos são utilizados, pois procuramos manter a correspondência com a terminologia apresentada nos artigos originais.

O estado atual define quais processos são considerados "bons", ou seja, que não tenham falhado. Assim o *líder* sabe quais *acceptors* estão ativos no protocolo de acordo e, consequentemente, para qual conjunto de *acceptors* ele deve enviar mensagens. De acordo com uma constante fixada c, definimos que os *acceptors* e *quóruns* usados pela instância i do algoritmo de consenso seja determinado pelo estado $i - c$. Antes de tomar qualquer ação para a instância i, um líder espera até que saiba o estado $i-c$ da máquina de estados. Em outras palavras, um líder deve esperar até que ele saiba todos os comandos até o comando de número $i - c$ antes que ele descubra para quais *acceptors* ele deve mandar as mensagens *propose* para a i-ésima instância do algoritmo de consenso Dynamic Paxos.

2.4. Cheap Paxos

No Paxos usual, para que seja possível tolerar f falhas no sistema, são necessários $2f + 1$ *acceptors* participando do algoritmo de consenso em todas as instâncias. O Cheap Paxos [Lamport and Massa 2004] propõe que o sistema rodando Paxos tolere f falhas se ele usar $f + 1$ *acceptors* principais e f *acceptors* auxiliares. Os *acceptors* auxiliares não precisam participar na escolha dos valores e só precisam agir no caso de um *acceptors* principal falhar e, portanto, podem ser menores, mais lentos, mais baratos, ou executarem em nodos primariamente focados em outras tarefas do sistema.

O Cheap Paxos é principalmente baseado na observação sobre o algoritmo Paxos de que um líder pode mandar suas mensagens apenas para um quórum de *acceptors* ao invés de enviar para todos os *acceptors* do sistema, e ainda assim chegar no consenso, contanto que todos os *acceptors* no quórum estejam funcionando e consigam se comunicar com o líder e os *learners*. Na ausência de falhas, não há necessidade que os *acceptors* que não fazem parte do quórum participem.

No Cheap Paxos, o Dynamic Paxos é usado para configurar o sistema de forma que o conjunto de *acceptors* principais que estão funcionando formem um quórum. Desde que esses *acceptors* continuem funcionando, eles garantem progresso ao sistema. Se um deles falhar, então o quórum dos *acceptors* principais não consegue mais escolher comandos. Um quórum diferente, contendo um ou mais *acceptors* auxiliares é usado para completar a execução de quaisquer instâncias do algoritmo de consenso Paxos que estavam em progresso quando a falha ocorreu. A reconfiguração remove o participante falho e modifica o conjunto de quóruns de forma que os *acceptors* principais restantes formem um quórum. Esses *acceptors* principais, podem então continuar a executar o sistema, e os processadores auxiliares voltam a ficar ociosos.

2.5. Flexible Paxos

O Flexible Paxos [Howard et al. 2016] percebe que o algoritmo do Paxos é mais conservador do que necessário, já que usa o mesmo quórum de maioria em ambas as *fases 1* e 2. Na realidade, é possível separar os quóruns das *fases 1* e 2 em diferentes quóruns, $Q1$ e $Q2$ e ainda manter a consistência do sistema. Além disso, existem diferentes formas de decidir um quórum que não necessariamente precisa ser de maioria, como por exemplo quóruns simples e *grid* quóruns.

Pela observação do Flexible Paxos, é necessário que os quóruns de $Q1$ e $Q2$ tenham interseção entre si, mas eles não precisam conter exatamente os mesmos *acceptors*, como acontece no Paxos tradicional. Com isso, o Flexible Paxos permite diferentes configurações de quóruns, como por exemplo, diminuir o tamanho de $Q2$ ao

custo de aumentar o tamanho de $Q1$. Como a *fase 2* é executada muitas mais vezes em implementações como o Multi Paxos, diminuir o $Q2$ pode trazer uma melhoria de desempenho.

2.6. Ring Paxos e Multi-Ring Paxos

O Ring Paxos [Marandi et al. 2010] procura reduzir a carga da comunicação no protocolo Paxos, substituindo a comunicação nodo a nodo do Paxos tradicional por *IP multicast* e utilizando uma topologia de anel lógico sobre o protocolo.

O protocolo opera de maneira muito similar ao Paxos tradicional, com a diferença de que um dos *proposers* vira líder e é chamado de coordenador, ficando responsável por receber as propostas dos outros *proposers* e propô-las aos *acceptors*. Para virar coordenador um *proposer* executa a *fase 1* do Paxos enviando um número de proposta através de uma mensagem *IP multicast* que também contém dados sobre a configuração do anel e qual *acceptor* é o inicio do mesmo. Cada *acceptor* responde individualmente ao *proposer* que quer virar coordenador e se o seu número de proposta for o maior já visto, o *acceptor* aceita o *proposer* como coordenador que então continua para a próxima parte do algoritmo assim que houver um quórum de *acceptors* que o reconhecem como líder.

Na *fase 2* do algoritmo, o coordenador envia novamente uma mensagem por *IP multicast* a todos os *acceptors*, desta vez contendo o valor a ser decidido acompanhado de um identificador do valor (chamado de $c - vid$ no algoritmo) que é armazenado pelos *acceptors* em seu identificador de valor próprio (chamado de $v - vid$ no algoritmo). Os *acceptors* não respondem individualmente ao coordenador, ao invés disso, o primeiro *acceptor* do anel manda sua resposta que também contém o $c - vid$ ao seu sucessor que faz o mesmo novamente se o $c - vid$ recebido corresponder com seu $v - vid$. Isso se repete até que a mensagem chegue ao último *acceptor* do anel, que encaminha as repostas ao coordenador. Se houver algum problema antes de chegar a este ponto, ou o $v - vid$ de algum dos *acceptors* for diferente do $c - vid$, o coordenador terá que tentar novamente com um número de proposta maior após um *timeout*.

O Multi-Ring Paxos [Marandi et al. 2012] expande sobre o Ring Paxos, ao usar múltiplas instâncias independentes do protocolo para escalar a vazão sem sacrificar o tempo de resposta. Desta forma, os *acceptors* são divididos em diferentes grupos, que formam anéis lógicos separados e que operam da mesma maneira descrita anteriormente.

2.7. PigPaxos

O PigPaxos [Charapko et al. 2021] substitui a comunicação direta entre o *proposer coordenador* e os *acceptors* por uma comunicação com fluxo de mensagens baseado em retransmissão e agregação, com o intuito de resolver o gargalo causado por um único proponente e melhorar a vazão dos protocolos de replicação fortemente consistentes.

Segundo os autores, a principal causa do gargalo observado quando há um único líder no protocolo é consequência de que muitas mensagens entram e saem de um único nodo, e também são processadas neste mesmo nodo, onde executa o líder (*proposer coordenador*). Este volume de mensagens leva comumente a saturação da CPU com serialização, deserialização e processamento de mensagens. Além disso, dependendo do tamanho das mensagens, pode haver saturação da largura de rede do líder.

Os autores propõem uma primitiva chamada *Pig* que modifica como a comunicação entre nodos opera. A primitiva *Pig* difunde mensagens para múltiplos destinatários e agrega as respostas dos destinatários. A primitiva evita a comunicação direta entre os nodos iniciante e destinatários. Ao invés disso, o nodo iniciante escolhe um nodo intermediário para desempenhar o papel de retransmissor. Este nodo retransmite mensagens de *proposers* para os nodos restantes, os seguidores, e agrega as respostas no caminho de volta ao nodo iniciante.

Com apoio da primitiva *Pig*, os *acceptors* são divididos em grupos de retransmissão não sobrepostos. Quando o líder inicia uma comunicação com os *acceptors*, ele escolhe aleatoriamente um retransmissor de cada grupo. Ao receber a mensagem do líder, um nodo retransmissor irá processá-la normalmente e retransmitir para os outros nodos do grupo. Ao receber a mensagem, os seguidores processam a mensagem e respondem ao nodo retransmissor como se ele fosse o líder. Cada nodo retransmissor espera as respostas dos seguidores, e agrega elas em uma única mensagem para responder ao líder. O nodo retransmissor espera por todos os seguidores antes de responder o líder, mas para lidar com o atraso de seguidores, um tempo de espera máximo é definido. O líder faz a agregação das repostas dos retransmissores.

2.8. Generalized Paxos

Em [Lamport 2005] Lamport generaliza o problema de acordo sobre um único valor para o acordo sobre um conjunto incremental de valores. A intuição baseia-se no modelo Multi-Paxos, onde uma sequência de valores deve ser acordada. A generalização proposta por Lamport observa que as instâncias não precisam ser decididas em ordem. Por exemplo, um *learner* poderia aprender o 5^o valor antes de aprender o 3^o, mesmo sabendo que a ordem para entrega (ou processamento) dos valores decididos deve ser crescente, ou seja, o 5^o comando só pode ser processado apos o 4^o comando.

Com essa observação, e acrescentando a cada *learner* l uma variável que mantém a lista de instâncias aprendidas, $learned[l]$, as seguintes propriedades são definidas:

- *Não-trivialidade*: para qualquer *learner* l, o valor de $learned[l]$ é sempre uma sequência dos valores propostos;
- *Estabilidade*: para qualquer *learner* l, o valor de $learned[l]$ em qualquer instante de tempo é um prefixo do seu valor em um instante futuro de tempo;
- *Consistência*: para quaisquer *learners* l_1 e l_2, sempre será o caso de que uma das sequências $learned[l_1]$ e $learned[l_2]$ é um prefixo da outra;
- *Vivacidade(c,l)*: se um comando c foi proposto, então eventualmente a sequência $learned[l]$ vai conter o elemento c.

Para atender as propriedades acima, é proposto o uso de uma estrutura chamada *command-structure*, ou $c - struct$, que auxilia a determinar se uma instância de consenso proposta conflita ou não com outras instâncias aprendidas pelos demais *learners* do sistema. Se duas instâncias não conflitam, a ordem relativa entre elas não é importante. Esta proposta permite a entrega de comandos em uma ordem parcial, sendo que os comandos conflitantes serão sempre entregues em uma mesma ordem relativa a todos os participantes do sistema.

É importante destacar que a noção de conflito entre comandos é dependente da semântica da aplicação. Por exemplo, duas operações de leitura, ex. $r(x)$ e $r(y)$ nunca

conflitam. Já uma operação de leitura e outra de escrita na mesma variável, conflitam. Por exemplo, $r(x)$ e $w(x, v)$ conflitam.

2.9. Egalitarian Paxos

Egalitarian Paxos [Moraru et al. 2013], ou EPaxos, apresenta uma abordagem com propostas de instâncias de consenso concorrentes. Este protocolo assume um modelo Multi-Paxos, onde uma sequência de valores é proposta, representados por comandos, de forma que cada *learner* recebe uma sequência de comandos e cuja execução da sequência de comandos é consistente entre réplicas. Cada nodo participante pode, de forma oportunista, tornar-se um líder para a proposta de algum comando (*i.e.*, uma instância de consenso). Quando um comando c não conflita com outros comandos concorrentes, este é efetivado em uma rodada após a recepção da confirmação de um quórum rápido de *acceptors*. Desta forma, o EPaxos acaba compactando a *fase 2*, tornando-a parte de *fase 1* quando não há conflitos entre o comando proposto e os demais comandos.

A presença ou ausência de conflitos entre instâncias pendentes é registrada por cada nodo participante em um conjunto de conflitos. Na *fase 1*, quando um líder envia uma instância a outros participantes, estes avaliam se a instância conflita com outras instâncias pendentes, e registra no seu conjunto de conflitos local, se for o caso. O líder reúne os conjuntos de conflito de diferentes participantes e se todos têm o mesmo conjunto, possivelmente vazio, significa que os participantes têm uma mesma visão e o caminho rápido pode ser tomado. Porém, se o quórum rápido detecta conflito entre comandos, EPaxos retorna para o modo do Paxos tradicional e prossegue com uma segunda fase para estabelecer a ordem entre os comandos conflitantes. A grande vantagem do EPaxos é conseguir decidir comandos não-conflitantes em apenas uma rodada de troca de mensagens.

2.10. Vertical Paxos

O *Vertical Paxos* [Lamport et al. 2009] permite a reconfiguração de *acceptors* durante a execução de uma instância do protocolo Paxos. Algumas variações do Paxos realizam reconfigurações de forma "horizontal", ou seja, as configurações apenas mudam de uma instância do protocolo Paxos para outra. Já no *Vertical Paxos*, as configurações mudam "verticalmente" durante a execução de uma instância e as configurações permanecem as mesmas quando há movimento horizontal, indo de um número de proposta em uma instância para o mesmo número de proposta em qualquer outra instância.

O *Vertical Paxos* faz uso de um mestre auxiliar de reconfiguração para permitir que a mudança de configurações altere os subconjuntos de *acceptors* em cada instância de consenso do algoritmo. O mestre determina o conjunto de *acceptors* e o líder para cada configuração. Para eliminar a dependência dos *acceptors* sobre números de propostas anteriores quando ocorre uma reconfiguração, existem duas abordagens descritas pelo algoritmo.

Na primeira abordagem, quando a configuração muda ela fica ativa instantaneamente e a configuração antiga apenas continua ativa para o armazenamento de informações passadas e assim que o estado da configuração anterior termina de ser transferido para a nova configuração, o novo líder informa o mestre de reconfiguração para que ele possa informar todos os futuros líderes que não precisam acessar a configuração

antiga. Isso traz um problema, já que o líder na nova configuração pode falhar antes que termine de transferir o estado da configuração antiga para a nova. Assim, o líder que substitui o líder falho deve acessar as configurações de todas números de propostas que não foram propriamente transferidas. Se ocorrerem falhas de líderes em sequência, isso pode fazer com que muitos acessos de configurações fiquem pendentes quando um novo líder cria uma nova configuração.

Para resolver isso, na segunda abordagem a nova configuração inicia inativa e apenas após a transferência da configuração anterior ser concluída e o mestre de reconfiguração ser avisado que a configuração nova pode ser ativada que ela passa a receber comandos. Desta forma, se o novo líder falhar antes que a nova configuração transfira o estado da configuração anterior, a nova configuração que falhou não será ativada e outras configurações que venham após ela não consideraram a configuração falha.

2.11. WPaxos

O WPaxos [Ailijiang et al. 2019] é uma variante do Paxos com múltiplos líderes, projetada para redes de longa distância, tirando vantagem de quóruns flexíveis do *Flexible Paxos* para melhorar o desempenho do algoritmo nestas redes, principalmente na presença de localidade de acesso.

Ao invés de usar quóruns flexíveis com um único líder, como acontece no *Flexible Paxos*, o WPaxos apresenta um protocolo com múltiplos líderes com quóruns flexíveis, onde cada nodo pode atuar como líder para um subconjunto de *acceptors* de forma independente através de uma relação de posse. Diferentemente do Vertical Paxos, o WPaxos não considera a mudança de posse sobre os *acceptors* como uma operação de reconfiguração. A troca de posse é realizada usando o próprio protocolo Paxos.

Os diferentes líderes "roubam" *acceptors* uns dos outros pela execução da *fase 1* sobre a rede de longa distância, que faz com que o número de proposta de diferentes *acceptors* aumente, mudando seus donos. Ao virar dono de um *acceptor*, o *proposer* repete a *fase 2* do algoritmo múltiplas vezes para decidir valores em diferentes instâncias, sem precisar repetir a *fase 1* enquanto não for necessário tomar posse de outros *acceptors*.

2.12. Outras variação de Paxos

Devido à limitação de espaço, restringimos o detalhamento de algumas das principais variações do protocolo. Porém, esta não é uma comparação exaustiva, outras propostas interessantes são mencionadas a seguir. O Kernel Paxos [Esposito et al. 2018] faz a implementação do protocolo Paxos a nível de núcleo do sistema operacional Linux com o intuito de reduzir a sobrecarga de comunicação do Paxos tradicional ao remover as trocas de contexto e o uso da pilha TCP/IP. O P4xos [Dang et al. 2020] implementa o Paxos diretamente em *switches* de rede, fazendo uso da linguagem de alto nível "P4" para definir o processamento de pacotes em *switches* programáveis. O CASPaxos [Rystsov 2018] propõe uma versão de protocolo de consenso sem a replicação de histórico de operações.

3. Análise e Otimizações no Protocolo Paxos

A despeito das particularidades de cada protocolo apresentado, uma implementação do protocolo Paxos segue, em geral, uma estrutura comum: Há um conjunto de *proposers*,

responsáveis pela proposta de valores; Um conjunto de *acceptors*, responsáveis por aceitar um valor único em cada instância de consenso; Um conjunto de *learners*, responsáveis por aprender os valores decididos pelo algoritmo; Uma sucessão de *fases*, que dividem o algoritmo em rodadas com diferentes propósitos (ex. preparação, proposta, disseminação do valor decidido); Um *quórum*, caracterizado por um subconjunto de *acceptors* que possuem interseção entre si em cada fase, para cada instância de consenso. Esse quórum, normalmente é representado pela maioria dos *acceptors* e garante que apenas um valor seja decidido.

Variantes do Paxos fazem otimizações conforme necessidades específicas, como aumento de vazão, redução de latência, redução no custo de hardware, dentre outros requisitos. A seguir nós avaliamos os aspectos modificados em relação ao algoritmo tradicional e quais as otimizações alcançadas devido a estas mudanças.

Fases e o volume de mensagens trocadas: Com relação ao número de fases, no Paxos tradicional existem a *fase 1*, iniciada pelos *proposers* para obter o comprometimento dos *acceptors* a participarem de uma decisão associada a um número de proposta; a *fase 2*, para envio do valor associado à proposta e, caso um quórum de *acceptors* confirme a aceitação do valor, ocorre a *fase 3*, onde o valor decidido é disseminado aos *learners*.

O Multi Paxos modifica o funcionamento das fases do algoritmo, permitindo que um *proposer* líder seja definido e possa fazer uma pré-reserva para um conjunto de n instâncias de consenso. Após a reserva, o líder realiza apenas a *fase 2* do algoritmo n vezes, diminuindo o número de mensagens que precisam ser trocadas. Outra otimização, permitindo a eliminação da *fase 3* (divulgação do valor decidido para os *learners*) é fazer com que os *acceptors* encaminhem os valores aceitos, ao final da *fase 2*, tanto ao *proposers* quanto aos *learners*. Assim, os *learners* podem concluir o valor decidido ao receber o mesmo valor de um quórum de *acceptors*. Esta otimização reduz o número de fases, mas aumento o número de mensagens trocadas, pois cada *acceptor* enviará o dobro de mensagens. Supondo configurações com um grande número de *acceptors*, esse volume de mensagens pode se tornar um gargalo.

O Fast Paxos adiciona um tipo de instância do algoritmo chamada de "rápida" que modifica o comportamento dos *acceptors* ao receberem uma mensagem que permite que aceitem diferentes valores independentemente dos demais *acceptors*. Isso reduz o número de mensagens necessárias para chegar no consenso uma vez que um *proposer* pode enviar mensagens diretamente para os *acceptors* mesmo sem ser o *proposer* líder.

O PigPaxos cria sub-fases de retransmissão e agregação de mensagens dentro das fases usuais do Paxos, designando *acceptors* retransmissores. Essa abordagem visa reduzir o gargalo da disseminação e agregação de muitas mensagens simultaneamente nos *proposers* ao custo de retransmissões e aumento no número de fases do protocolo. Essa abordagem é indicada em sistemas onde há congestionamento do *proposer*, porém pode-se esperar um aumento na latência do protocolo.

Composição e tamanho do quórum de acceptors: O Dynamic Paxos permite a mudança de quóruns em tempo de execução ao reconfigurar uma máquina de estados que define quais *acceptors* estão ativos e participando do quórum. A abordagem permite maior controle sobre os quóruns ativos no sistema ao determinar quais *acceptors* estão ativos em caso de falhas.

O Cheap Paxos expande o Dynamic Paxos, usando quóruns definidos de forma dinâmica para separar o sistema em *acceptors* principais, que fazem parte do quórum do consenso, e *acceptors* secundários, com desempenho inferior, que a principio não fazem parte do quórum. A principal ideia do Cheap Paxos é reduzir custos com *acceptors* de alto desempenho, permitindo que *acceptors* auxiliares, que apenas fazem parte do quórum quando há alguma falha em um *acceptor* principal, executem em máquinas mais baratas. Apesar de não ser prevista na época, essa abordagem pode ser atrativa em ambientes com alocação de infraestrutura sob-demanda, como infraestruturas de computação em nuvem, otimizando o uso de recursos e reduzindo gastos com equipamentos.

O Flexible Paxos por outro lado, faz alterações em como os quóruns funcionam, permitindo que os quóruns de diferentes fases tenham tamanhos variados, desde que possuam interseção entre si. Assim, é possível diminuir o tamanho do quórum de uma fase enquanto se aumenta o quórum de outra fase, o que pode levar a melhoras de desempenho se o quórum em fases executadas mais frequentemente for reduzido, com um menor número de *acceptors* participantes.

O Vertical Paxos introduz um nodo coordenador de configurações responsável pela configuração de *acceptors* ativos nos quóruns e o respectivo *proposer* líder para cada número de proposta que reflete em todas as instâncias de consenso. A possibilidade de mudar as configurações dos *acceptors* é útil em banco de dados geo-replicados por permitir a realocação, atribuição de dados ou objetos à diferentes nodos líderes, permitindo que o sistema se ajuste à mudanças na localidade de acesso. O WPaxos, da mesma forma que o *Flexible Paxos*, usa a ideia de quóruns flexíveis, mas também altera o funcionamento dos *Proposers* e *Acceptors*, permitindo múltiplos líderes ao mesmo tempo para diferentes grupos de *Acceptors*. Os líderes podem tomar posse de *Acceptors* de outros líderes, visando também melhorias quanto a localidade de acesso.

Paralelização na decisão de valores: O Generalized Paxos oferece maior liberdade na execução de instâncias do Paxos, permitindo decidir instâncias independentes sem necessitar de um ordenamento total entre elas. Essa abordagem relaxa o critério de ordenação total entre os valores decididos, visto que não é necessário aguardar instâncias anteriores para processar uma nova instância que não dependa das instâncias antigas. Observa-se que *learners* podem aprender valores em sequências de consenso com ordenações parciais. Requisições dependentes estarão dispostas em uma mesma ordem relativa na sequência de valores e requisições independentes podem aparecer em diferentes posições da sequência.

O Egalitarian Paxos também oferece concorrência entre instâncias. Ele modifica o funcionamento das fases do algoritmo compactando as *fases 1 e 2* enquanto não houver conflitos entre as propostas. O protocolo implementa uma abordagem oportunista para a proposta de valores onde qualquer nodo pode ser um *proposer*. Múltiplas instâncias de consenso podem estar pendentes em um determinado momento sem causar problemas desde que não exista conflitos entre os valores propostos em cada instância. Em cenários com poucos conflitos, o Egalitarian Paxos alcança uma melhora na latência em redes de longa distância, e equilíbrio de carga entre réplicas, aumentando a vazão do protocolo. Comparado com o Generalized Paxos, a grande vantagem do Egalitarian Paxos é conseguir decidir comandos não-conflitantes em apenas uma rodada de troca de mensagens. Ele funciona bem quando há operações sobre inúmeros objetos e a probabilidade de quaisquer

dois nodos operando no mesmo objeto é muito baixa. Esse tipo de protocolo é vantajoso para implementar replicação em *multi-datacenters* com requisitos de consistência forte.

O Multi-Ring Paxos também explora concorrência na entrega de valores decididos em instâncias não conflitantes. Nesse protocolo, diferentes interfaces de rede podem ser usadas para rodar instâncias independentes, levando não apenas a um aumento de vazão do protocolo, mas também um aumento na capacidade de rede, desde que nodos estejam equipados com mais de uma interface de rede.

4. Discussão

Neste artigo foi feito um estudo do protocolo Paxos e suas principais variações, destacando o funcionamento das variantes do Paxos e e as otimizações de cada versão. Ao avaliar as otimizações propostas, é possível perceber tendências sobre quais variantes do Paxos melhor atendem requisitos de aplicações com contextos ou tecnologias específicas.

Por exemplo, plataformas de computação em nuvem ou névoa combinam compartilhamento de recursos e a necessidade de realocações e migrações. Protocolos Paxos que melhor se adaptam a estas plataformas devem possuir agilidade na recuperação e reconfiguração de grupos de participantes, como se observa no Dynamic Paxos, Vertical Paxos e WPaxos. Além disso, o custo pelo uso de recursos nestas plataformas varia em função do tipo de máquina utilizada. O Cheap Paxos visa reduzir o número de nodos de alto desempenho sem sacrificar completamente a tolerância a falhas, sendo uma alternativa interessante nesse contexto. Aplicações com grande número de participantes, como *Blockchains* ou *IoT* apresentam requisitos de escalabilidade. Variantes do Paxos com boas perspectivas de uso para estes cenários são o PigPaxos, *Egalitarian Paxos* e WPaxos.

Um desafio na comparação entre diferentes variantes do Paxos é a ausência de uma implementação em base de código comum, onde cada variante poderia alternar apenas detalhes referentes às otimizações da própria versão de Paxos. Como trabalho futuro, pretende-se desenvolver uma biblioteca de Paxos personalizável, onde as diferenças de cada variação possam ser empregadas e avaliadas. Além disso, uma biblioteca extensível também servirá como um artefato para testes e avaliação de novas versões do protocolo.

Referências

Ailijiang, A., Charapko, A., Demirbas, M., and Kosar, T. (2019). Wpaxos: Wide area network flexible consensus. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):211–223.

Chandra, T. D., Griesemer, R., and Redstone, J. (2007). Paxos made live: An engineering perspective. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, page 398–407.

Charapko, A., Ailijiang, A., and Demirbas, M. (2021). Pigpaxos: Devouring the communication bottlenecks in distributed consensus. In *Proceedings of the 2021 International Conference on Management of Data*, pages 235–247.

Dang, H. T., Bressana, P., Wang, H., Lee, K. S., Zilberman, N., Weatherspoon, H., Canini, M., Pedone, F., and Soulé, R. (2020). P4xos: Consensus as a network service. *IEEE/ACM Transactions on Networking*, 28(4):1726–1738.

Esposito, E. G., Coelho, P., and Pedone, F. (2018). Kernel paxos. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 231–240. IEEE.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382.

Howard, H., Malkhi, D., and Spiegelman, A. (2016). Flexible paxos: Quorum intersection revisited. *arXiv preprint arXiv:1608.06696*.

Junqueira, F. P., Reed, B. C., and Serafini, M. (2011). Zab: High-performance broadcast for primary-backup systems. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 245–256. IEEE.

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169.

Lamport, L. (2001). Paxos made simple. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 32(4):18–25.

Lamport, L. (2005). Generalized consensus and paxos. Technical Report MSR-TR-2005-33, Microsoft Research.

Lamport, L. (2006). Fast paxos. *Distributed Computing*, 19(2):79–103.

Lamport, L., Malkhi, D., and Zhou, L. (2009). Vertical paxos and primary-backup replication. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 312–313.

Lamport, L. and Massa, M. (2004). Cheap paxos. In *International Conference on Dependable Systems and Networks, 2004*, pages 307–314. IEEE.

Marandi, P. J., Primi, M., and Pedone, F. (2012). Multi-ring paxos. In *DSN*.

Marandi, P. J., Primi, M., Schiper, N., and Pedone, F. (2010). Ring paxos: A high-throughput atomic broadcast protocol. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 527–536. IEEE.

Moraru, I., Andersen, D. G., and Kaminsky, M. (2013). There is More Consensus in Egalitarian Parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13.

Oki, B. and Liskov, B. (1988). Viewstamped Replication: A general primary-copy method to support highly-available distributed systems. In *PODC*.

Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *USENIX ATC 2014*.

Pedone, F. and Schiper, A. (1999). Generic broadcast. In *Proceedings of the 13th International Symposium on Distributed Computing (DISC'99, formerly WDAG)*.

Rystsov, D. (2018). CASPaxos: Replicated State Machines without logs. *arXiv e-prints*, page arXiv:1802.07000.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM CSUR 1990*.