Reconfiguração Dinâmica de Protocolos Tolerantes a Faltas Bizantinas Baseados em Árvores

Tomás Pereira, Luís Rodrigues e Miguel Matos
{tomas.araujo.pereira, ler,
miguel.marques.matos}@tecnico.ulisboa.pt

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

Abstract. O uso de árvores de disseminação e agregação permite aumentar a escalabilidade e desempenho de protocolos de consenso tolerantes a faltas bizantinas. Infelizmente, a reconfiguração destas árvores pode ser complexa, o que leva a que muitos protocolos baseados em árvores usem um líder estável, o que nem sempre é desejável, por exemplo, por questões de equidade ou censura de transações. Neste artigo propomos técnicas eficientes para executar a mudança de líder e reconfigurar dinamicamente as árvores utilizadas pelo algoritmo de consenso. Através de uma avaliação experimental das técnicas propostas, mostramos que a reconfiguração dinâmica pode ser concretizada sem incorrer numa penalização significativa do desempenho.

Palavras-chave — Sistemas Distribuídos, Blockchain, Consenso Bizantino

1 Introdução

Os algoritmos de consenso Bizantino permitem que processos corretos cheguem a acordo mesmo na presença de (uma fração de) processos maliciosos. A importância destes algoritmos tem vindo a aumentar em parte pelo seu papel crucial no contexto de *blockchains*. Estes algoritmos obrigam à execução de várias rondas de troca de mensagens entre os participantes, pelo que as suas concretizações consomem uma quantidade significativa de recursos de comunicação e processamento. Desta forma, a capacidade de escala de muitas concretizações é limitada, restringindo a sua aplicação a sistemas com um número reduzido de participantes [1]. No entanto, existem aplicações de blockchain que necessitam de suportar números de participantes elevados (na ordem das centenas), o que obriga a desenvolver técnicas que permitam aumentar a capacidade de escala destes algoritmos.

O padrão de comunicação entre os participantes é um dos fatores chave na capacidade de escala do algoritmo de consenso. Protocolos como o PBFT [2] usam um padrão de comunicação *Todos-para-Todos*, com rondas onde todos os participantes enviam mensagens para todos os outros participantes. Protocolos como o

HotStuff [3] diminuem a complexidade da comunicação ao usar um padrão *Um-para-Todos Todos-para-Um*, em que um participante é escolhido como líder para agregar a informação enviada pelos restantes nós e, posteriormente, difundir a agregação pelos participantes. Esta abordagem reduz o custo de mensagens (de quadrático para linear), mas obriga o líder a receber, a processar e a enviar mensagens para todos os outros nós, podendo ser um ponto de estrangulamento do sistema. Em alternativa, protocolos como o Kauri [4] utilizam árvores de disseminação e agregação, onde a carga de trabalho do líder pode ser distribuída pelos nós interiores da árvore, o que permite aumentar substancialmente a escala do sistema. Uma desvantagem desta abordagem é que a latência para uma fase de consenso pode ser maior devido à necessidade de vários saltos entre o nó raiz e as folhas da árvore. Para compensar este efeito, o Kauri utiliza extensivamente técnicas de propagação e agregação *em encadeamento* (pipeline) para obter um bom desempenho.

Contudo, a reconfiguração de uma árvore de comunicação é complexa pois obriga à coordenação de todos o nós. Para além disso, a reconfiguração pode também afetar os mecanismos de execução em encadeamento, podendo levar a interrupções transitórias no fluxo de comunicação. Talvez por esta razão, praticamente todos os algoritmos de consenso baseados em árvore utilizam uma política de *líder estável* [4, 5, 6], em que uma mesma árvore é usada para várias instâncias consecutivas do consenso, realizando-se apenas uma reconfiguração quando existem suspeitas de que a árvore deixa ser robusta (por exemplo, quando a raiz da árvore, que corresponde ao líder, é suspeita de ter falhado). No entanto, no contexto de blockchains, existem vantagens em mudar de líder com frequência (e portanto, reconfigurar a árvore dinamicamente), por exemplo, para impedir que um líder malicioso censure certas transações, e por uma questão de equidade na produção de blocos.

Neste trabalho estudamos e avaliamos algoritmos para suportar a reconfiguração proactiva de árvores de disseminação e agregação em protocolos de consenso Bizantino. Para acelerar o processo de reconfiguração, os nossos algoritmos permitem que uma nova instância de consenso comece a usar uma dada árvore enquanto instâncias anteriores terminam a sua execução numa outra árvore. Esta abordagem permite execuções em que um nó pode começar a receber mensagens para uma nova instância de consenso sem ter ainda recebido mensagens referentes a instâncias anteriores (e que estão no passado causal da nova mensagem). O algoritmo proposto inclui mecanismos para lidar com estes cenários. Concretizámos o nosso algoritmo como uma extensão ao Kauri e apresentamos uma avaliação do mesmo baseada em código real. Talvez surpreendentemente, os nossos resultados experimentais mostram que a utilização de árvores de disseminação e agregação não é incompatível com políticas que pressupõem mudanças frequentes de líder.

2 Trabalho Relacionado

Existem na literatura diversos sistemas que usam árvores de disseminação e agregação para concretizar o acordo Bizantino. Entre estes salientamos os seguintes algoritmos: i) **Kauri** [4], algoritmo previamente mencionado pelo seu uso de árvores para aumentar a escala do sistema e evitar pontos de estrangulamento presentes em algoritmos com padrões de comunicação *Todos-para-Um*; ii) **ByzCoin** [5], que é um algoritmo desenhado para ser aplicado a *blockchains* à base de Prova-de-Trabalho, onde árvores são selecionadas por participantes com maior poder computacional. Na presença de falhas, a árvore rapidamente degenera numa topologia de *clique*, que tem um custo comunicacional elevado de $O(n^2)$; iii) **Motor** [7], que é uma extensão de ByzCoin onde complementa-se o algoritmo com árvores mais resilientes a falhas e com uma rotação de líderes para promover equidade de transações. Em comparação com o Kauri, este protocolo não mitiga a latência levantada pelo uso de árvores no consenso e, adicionalmente, em caso de falhas no líder, paga um custo desnecessário em rotações nas sub-árvores.

A grande vantagem dos algoritmos baseados em árvore é que ao distribuírem a carga possibilitam a paralelização do processo de disseminação e agregação, o que aumenta a capacidade de escala dos algoritmos. Possuem, no entanto, também algumas desvantagens. A profundidade da árvore pode não só aumentar a latência do processo de disseminação e agregação mas também gerar execuções onde alguns do nós estão inativos enquanto esperam que a informação se propague noutros ramos. Por outro lado, o número de árvores possíveis é exponencial e, caso uma árvore falhe, pode não ser trivial encontrar em tempo útil uma árvore capaz de suportar a execução do algoritmo.

O Kauri introduz mecanismos que resolvem os problemas atrás referidos. Para mitigar o efeito negativo que a latência de propagação na árvore pode ter no débito do sistema, o Kauri permite a execução de consenso *em encadeamento*, sendo que o sistema inicia a difusão otimista de um novo bloco mesmo antes da disseminação do bloco anterior estar concluída. Para garantir que, em caso de falha, é possível encontrar em tempo útil uma árvore que permita terminar o consenso, o Kauri propôs uma caracterização precisa desta propriedade (que se designa por *árvore robusta*), assim como um algoritmo para criar um sequência de árvores, das quais um subconjunto configurável é garantidamente robusto.

Tanto quanto é do nosso conhecimento, a maior parte dos sistemas que usam árvores de disseminação apenas reconfiguram a árvore quando se suspeita que esta deixou de ser robusta. Ou seja, a reconfiguração é reactiva, só ocorrendo quando o sistema já não apresenta um desempenho satisfatório (ou está mesmo bloqueado), com a finalidade de repor o nível de desempenho inicial. Como é evidente, neste cenário, o processo de reconfiguração só acarreta vantagens. No entanto, pode ser interessante reconfigurar a árvore proactivamente mesmo na ausência de falhas pois isto não só permite distribuir a carga pelos vários nós do sistema ao longo do tempo (uma vez que numa árvore, os nós interiores têm mais trabalho que os nós folha), como evita que um mesmo líder possa censurar certas transações por um período alargado. A eficiência do processo de reconfiguração

é fundamental, de forma a que a reconfiguração não se torne um fator limitativo do desempenho do sistema. Um sistema de consenso em árvore conhecido que utilize uma rotação de líderes por ronda é o Motor, mas é importante notar que vem de um contexto onde o custo de reconfiguração é reduzido quando comparado a sistemas como o Kauri, pois o Motor não usufrui de uma execução em encadeamento para um maior desempenho nem de uma construção de árvores mais complexa para garantir robustez. Neste trabalho, procuramos combinar as vantagens dos dois sistemas.

3 Reconfiguração Dinâmica

Nesta secção apresentamos um algoritmo para realizar a reconfiguração dinâmica de protocolos de acordo Bizantino que usam árvores de disseminação/agregação.

Um algoritmo de reconfiguração necessita de abordar os seguintes desafios:

- lidar com reconfigurações planeadas e reconfigurações forçadas (isto é, reconfigurações que são motivadas por falhas) de forma integrada.
- interferir o mínimo possível no débito do sistema. Em particular, o algoritmo deve evitar disromper os mecanismos de propagação em encadeamento usado por sistemas como o Kauri.
- lidar com situações em que a reconfiguração pode levar a que os participantes recebam mensagens por ordens que violam a causalidade.

Como será discutido posteriormente, o desempenho do algoritmo de reconfiguração poderá depender de vários fatores tais como a profundidade e grau da árvore, as diferenças entre a árvore de origem e a árvore de destino (trocar a posição de dois nós folha pode ter um efeito distinto do que trocar colocar na raiz um nó que anteriormente era uma folha), entre outros. Um dos objetivos deste trabalho é avaliar experimentalmente como é que estes fatores afetam o desempenho do nosso algoritmo. Na Secção 4, com a avaliação experimental conduzida, obtivemos resultados que verificam que fatores como árvores de profundidades diferentes, agendamentos de árvores diferentes e frequências de reconfiguração diferentes têm influência no impacto que a reconfiguração traz ao débito do sistema.

3.1 Pressupostos

Consideramos um protocolo para concretizar serviços de blockchain com permissão, isto é, onde o conjunto de participantes é conhecido. Dado este ambiente, assumimos que é fornecido a todos os participantes o agendamento de árvores (i.e., o escalonamento das várias configurações) e o fator de encadeamento de cada uma das configurações (i.e., o número de instâncias de consenso que podem ser iniciadas de forma optimista enquanto a atual ainda não concluiu).

Consideramos também que o sistema é tolerante a falhas Bizantinas, onde conseguimos suportar $f < \lfloor \frac{N-1}{3} \rfloor$ nós com comportamento arbitrário num total

de N nós. A única restrição imposta ao nós Bizantinos é não terem capacidade de comprometer as primitivas criptográficas. O sistema executa numa rede inevitavelmente síncrona, onde é possível garantir períodos de sincronia entre os participantes (só não se sabe quando) tal que possa haver progresso no sistema. Períodos de assincronia não possibilitam a violação da segurança do sistema.

Neste contexto, focamo-nos em protocolos determinísticos de consenso tolerantes a faltas Bizantinas baseados em líder. O líder comunica com os restantes nós recorrendo a uma árvore de disseminação e agregação da qual o líder é o nó raiz. O protocolo executa várias instâncias de consenso, em sequência. Assumimos que o protocolo encadeia estas execuções de forma otimista, podendo começar uma instância antes da anterior ter terminado. Cada instância necessita de várias rondas de comunicação para terminar.

Uma instância ao ser iniciada por um líder usa uma dada árvore para disseminar um bloco. Denominamos essa árvore como sendo a *configuração inicial* associada a essa instância. Assumimos que, na ausência de falhas, a configuração inicial para todas as instâncias está pré-definida. Ou seja, assumimos que existe um escalonamento pré-definido e globalmente conhecido do qual se obtém as árvores que vão ser usadas para executar a primeira ronda de cada instância.

Se, na ausência de falhas, todas as instâncias usarem a mesma configuração, consideramos que o protocolo usa uma política de *líder estável*. Se nem todas as instâncias usarem a mesma configuração, dizemos que o protocolo suporta uma *reconfiguração dinâmica*. A reconfiguração dinâmica pode ocorrer sempre que se inicia uma nova instância ou periodicamente. Além disso as várias rondas de uma dada instância podem usar todas a mesma configuração para fazer a disseminação e agregação de valores ou, alternativamente, usar configurações diferentes. No caso do Kauri, na ausência de falhas, uma instância usa a mesma configuração de árvore para todas as rondas.

O protocolo assume o mesmo comportamento que o Kauri em relação a falhas Bizantinas, sendo que o novo processo de reconfiguração e a política de líder rotativo não alteram nem afetam os mecanismos de recuperação do protocolo original, em particular, nas execuções onde ocorrem períodos de assincronia ou nas execuções em que nós maliciosos tentam atrasar a troca de configurações. Um nó, ao detetar a ausência de progresso por um período definido de tempo, avançará para a próxima configuração do escalonamento.

3.2 Agendamento de Árvores

Um agendamento de árvores dita a sequência de árvores diferentes que serão utilizadas ao longo de uma execução. Estas são identificadas de 0 a n, sendo n um número arbitrário. A partir de uma árvore, qualquer nó conseguirá inferir a raiz da árvore (i.e., o líder), o seu pai (caso tenha) e os seus filhos (caso existam). Adicionalmente, cada árvore poderá ter valores distintos para o grau dos nós (i.e, o leque de filhos que cada nó interno tem) e para o tamanho da profundidade do encadeamento, sendo estes atributos dinâmicos ao longo da execução do sistema. Por último, cada árvore tem uma *duração alvo*, caracterizada pelo número de blocos em que a árvore estará em vigor até reconfigurar.

Como simplificação, assumiremos que todas as árvores têm um alvo igual de k blocos, embora esta propriedade também possa ser configurada de forma distinta de árvore para árvore. Ao atingir o alvo na árvore n, o sistema irá reconfigurar e retomar consenso com a árvore 0, resultando num agendamento cíclico. No protótipo atual, em caso de falhas, as configurações do agendamento não são recalculadas.

Para garantir equidade no sistema, é desejável utilizar um agendamento de pelo menos N árvores diferentes em que cada uma delas tenha um participante diferente como raiz. Obviamente, é possível considerar algoritmos de construção de (sequências de) árvores mais complexos, que tenham em conta a capacidade computacional e a localização dos nós, assim como as propriedades das ligações da rede (latência, débito, etc). Esses algoritmos são complementares às contribuições deste artigo, tendo algum trabalho preliminar nessa direção sido apresentado em [8].

3.3 Transição entre Árvores

Um dos maiores desafios deste trabalho é reduzir o custo de reconfiguração. Este custo tem duas causas principais: a latência originada pela profundidade da árvore e o processamento dos blocos em transição no encadeamento de instâncias do sistema. Isto porque nós nos níveis superiores receberão informação mais cedo do que os nós nos níveis inferiores, logo o tempo que demorará ao sistema a começar a propor blocos novos poderá ser dependente do nível onde se encontra o próximo líder. Esta lógica acaba por se aplicar à árvore na sua completude: quantos mais nós dos níveis de baixo subirem na hierarquia da árvore na próxima configuração, maior é a probabilidade do sistema ter que esperar pelos mesmos.

Sabe-se então que para qualquer configuração $i \geq 0$ são submetidos k blocos tal que B_{ik+j}, onde $1 \leq j \leq k$, identifique um bloco nessa configuração. Para compensar pela espera mencionada acima, paralelizamos a transição entre duas árvores com a seguinte otimização: sabendo que o bloco B_{ik+k} é o último bloco da árvore T_i, o líder dessa árvore irá transicionar para a configuração nova assim que propuser o bloco. Mesmo assim, o bloco B_{ik+k} será terminado na árvore em que foi proposto, ou seja, na árvore T_i. Todos os nós que receberem o bloco B_{ik+k} também transicionarão para a árvore nova assim que verificarem que o bloco proposto é válido e votarem. Destes nós, o que for líder da árvore T_{i+1} poderá começar a propor blocos dentro da nova configuração assim que acabar de reconfigurar. Assim sendo, o sistema irá finalizar o encadeamento da configuração anterior e em paralelo encher o encadeamento da configuração nova. Este processo é facilitado pelo facto que as mensagens do protocolo incluem o identificador da árvore, permitindo ao sistema que duas árvores sejam utilizadas em simultâneo e que nós consigam diferenciar os destinatários para tal.

Um nó que receba o bloco B_{ik+k} mas que ainda tenha blocos no passado causal de B_{ik+k} por receber (i.e., propostas de quaisquer instância de consenso j, onde $ik < j < ik + k$, por receber) só se juntará à configuração nova quando estes forem recebidos e processados. Só a partir desse momento é que o nó poderá processar B_{ik+k} e subsequentemente reconfigurar para participar no consenso da

nova árvore. Isto porque um nó para poder reconfigurar, para além de precisar de testemunhar e votar na proposta que concretiza o alvo da configuração em que está inserido, precisa de primeiro receber e votar em todas as propostas no passado causal do bloco B_{ik+k}.

Após a reconfiguração, o nó processa as propostas pendentes por ordem e envia-as para os seus filhos dentro da nova configuração. Estes casos de fronteira são exemplificados na Figura 1.

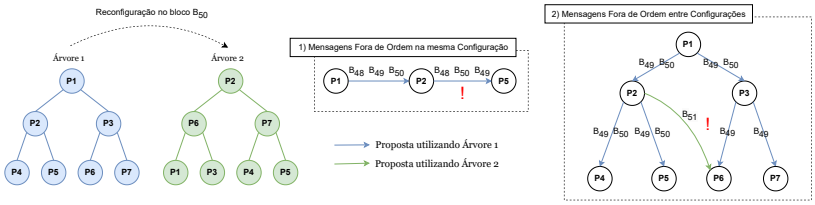

Figura 1: Alguns casos a ter em consideração: 1) Nós que possam receber a proposta que atinge o alvo de uma árvore fora de ordem. 2) Nós que possam receber propostas de uma configuração futura antes de reconfigurarem.

Dito isto, esperamos que seja favorável para o sistema agendar árvores de forma a que pares de árvores consecutivas diferenciem pouco nas suas arestas. Ao manter estruturas de encadeamento semelhantes, o sistema incorrerá menos esperas por nós que se encontravam numa profundidade mais baixa da árvore inicial.

3.4 Vantagens de Agendamentos Dinâmicos

Dado o contexto do problema, uma política de líder rotativo em sistemas em árvore oferece uma flexibilidade adicional ao permitir mudar também a estrutura da árvore, o que não acontece em sistemas que usam uma estrela. O número de árvores possíveis é exponencial e, portanto, exercer uma escolha criteriosa destas árvores é crucial. Considerando que o protocolo será aplicado num contexto de blockchains com permissão, podemos assumir que os nós poderão obter uma estimativa da qualidade das ligações entre si. Logo, o agendamento de árvores poderá utilizar esta informação para definir árvores que conseguem obter um desempenho melhor em expectativa, quando comparado a árvores aleatórias numa rede geo-distribuída. Para além disso, será necessário adaptar o fator de encadeamento para maximizar o desempenho de cada árvore.

Como base para a avaliação experimental, de forma a garantir um agendamento que permita que todos os nós assumam o papel de líder (ou seja, sejam raiz de uma árvore), assumiremos que as execuções utilizarão um agendamento baseado na rotação dos nós da árvore. Isto é, cada árvore prevista no agendamento é obtida a partir da árvore anterior, através da rotação dos nós.

4 Avaliação

Nesta secção, apresentaremos vários casos de interesse que permitem inferir o impacto de uma rotação de árvores no Kauri. Esta avaliação foi feita recorrendo a uma concretização preliminar da nossa proposta. Para testarmos diferentes condições de rede recorremos ao Kollaps [9], que permite emular as diferentes características de uma rede distribuída, tais como a latência e a largura de rede entre cada par de nós. Todas as experiências foram realizadas usando uma única máquina física com recursos computacionais suficientes para evitar entrar em sobrecarga. Recorrendo ao Kollaps, controlámos a latência e largura de banda da rede. Em todas as experiências atribuímos as mesmas características de rede a todas as ligações: uma latência de *50 ms* e uma largura de banda de *750 Kbp/s*. Embora o nosso sistema tenha como objetivo final ser aplicável a redes heterogéneas de larga escala, neste artigo optámos por fazer a avaliação usando apenas redes homogéneas. Desta forma, a única fonte de variações no desempenho são os mecanismos que pretendemos avaliar, o que simplifica a análise.

Todas as figuras usam a seguinte notação. O débito do sistema é medido usando o valor médio do número de blocos decididos por segundo ao longo de um dado período. Quando ocorre uma reconfiguração, o momento em que esta é despoletada é realçado com um diamante. Uma vez que as reconfigurações ocorrem a cada k blocos, os sistemas com maior débito acabam por reconfigurar mais frequentemente.

Sendo assim, estudamos o impacto dos seguintes três fatores no desempenho do sistema: i) a profundidade da árvore; ii) a deslocação de nós entre configurações e iii) a frequência da reconfiguração.

4.1 Profundidade da Árvore

Nesta secção avaliamos como a profundidade da árvore afeta a reconfiguração. Em todos os casos, as árvores são reconfiguradas a cada $k = 100$ blocos. Avaliámos os seguintes cenários:

(a) Profundidade $h = 2$: usando $N = 43$ nós organizados numa estrela de grau $m = 42$. Este cenário corresponde ao HotStuff [3].
(b) Profundidade $h = 3$: usando $N = 43$ nós organizados numa árvore de grau $m = 6$.
(c) Profundidade $h = 4$: usando $N = 40$ nós organizados numa árvore de grau $m = 3$.

Note-se que optámos por usar sempre árvores perfeitamente balanceadas, daí a ligeira diferença no número total de nós no último cenário. Nestes cenários, a largura de banda disponível para cada nó é um ponto de estrangulamento quando o grau é grande. Por isso, a execução com árvores em formato de estrela é o cenário que tem pior desempenho, sendo que os outros dois cenários apresentam um débito semelhante, embora, como é natural, o cenário com maior profundidade apresente uma latência maior (não representada na figura).

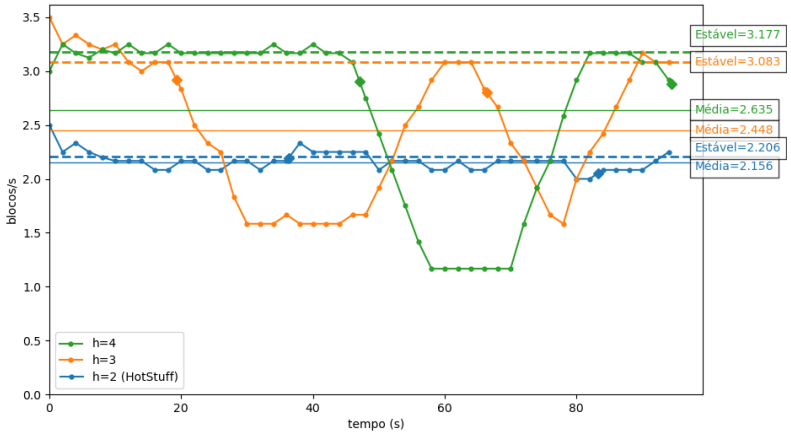

Figura 2: Débito (em blocos/s) ao longo do tempo em três cenários com profundidades diferentes.

O resultados das experiências são apresentados na Figura 2. A figura apresenta a evolução do débito de cada cenário ao longo do tempo, de forma a ilustrar o impacto da reconfiguração no mesmo. Adicionalmente, para cada cenário, apresentamos também o débito numa configuração de líder estável (linha a tracejado "Estável") para comparar com o valor médio da configuração proativa no gráfico (linha "Média").

A partir dos dados obtidos, podemos extrair as seguintes conclusões.

(a) No sistema que usa uma estrela a reconfiguração tem um impacto reduzido. Estes resultado não é surpreendente e está alinhado com os resultados conhecidos para sistemas como o HotStuff.

(b) O sistema que utiliza árvores de profundidade $h = 3$ revela um impacto menor no débito após reconfiguração quando comparado ao sistema que utiliza árvores de $h = 4$. Isto sugere que a reconfiguração de árvores mais profundas é mais lenta. Isto deve-se sobretudo ao facto de, no protótipo atual, os nós só começarem a participar na nova configuração depois de terem recebido informação de todos os blocos da configuração anterior, um processo que exibe maior latência para árvores mais profundas.

4.2 Deslocação de Nós

Nesta secção avaliamos o impacto que a localização dos nós em árvores adjacentes pode ter na reconfiguração. Usamos novamente o sistema com árvores de $N = 43$ nós e grau de $m = 6$. A reconfiguração continua a ser feita a cada $k = 100$

blocos. Comparamos o desempenho deste sistema usando duas estratégias de reconfiguração distintas, nomeadamente:

(a) Agendamento do tipo A. Este agendamento oscila entre duas árvores, onde a raiz da árvore seguinte é um dos filhos da raiz da árvore anterior. É esperado que este agendamento tenha um impacto reduzido na reconfiguração.
(b) Agendamento do tipo B. Neste agendamento, os nós interiores de uma árvore são nós folhas na árvore seguinte. Desta forma, a raiz de uma árvore é sempre substituída por um nó folha. É esperado que este agendamento tenha um impacto acrescido na reconfiguração.

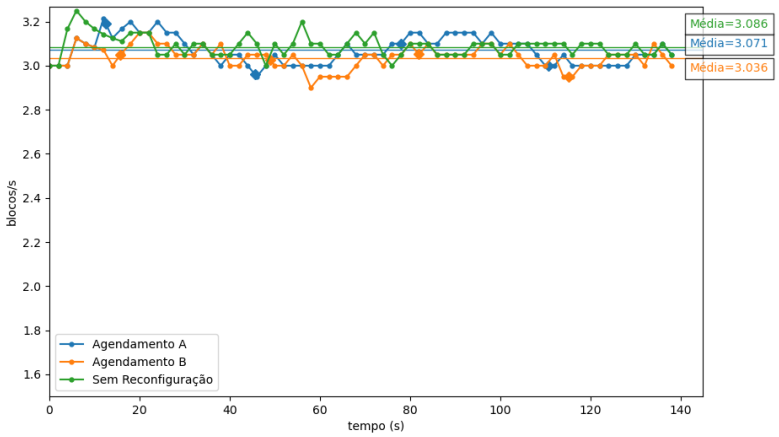

Figura 3: Débito (em blocos/s) ao longo do tempo entre dois sistemas com agendamentos diferentes, quando comparados a um sistema sem reconfiguração.

Os resultados são apresentados na Figura 3, adicionalmente com o caso de um sistema sem reconfiguração para comparação.

Talvez um pouco surpreendentemente, ambas as estratégias têm resultados praticamente iguais. Era esperado que o impacto da reconfiguração na execução com o agendamento A fosse menor do que no caso em que se usa o agendamento B, visto que ao trocar a raiz por um dos seus filhos a latência esperada é minimizada. No entanto, a execução com o agendamento B acaba por ter um débito bastante semelhante. Isto sugere que a localização nos nós nas árvores associadas a configurações consecutivas terá que ter em consideração a latência imposta pela limitação acima referida. Esta limitação é mitigada no agendamento de tipo B, pois este utiliza transições onde todos os nós interiores de uma

árvore nova pertencem à mesma camada da árvore da configuração anterior, algo que não acontece num agendamento rotativo, caso onde se notou especialmente o impacto da latência da reconfiguração (Figura 2).

4.3 Frequência da Reconfiguração

Finalmente, estudamos o impacto da frequência da reconfiguração no desempenho do sistema. Voltamos a utilizar um agendamento rotativo e continuamos a usar o mesmo sistema de $N = 43$ nós e grau $m = 6$, avaliando desta vez o desempenho para quatro valores diferentes de k, nomeadamente:

(a) $k = 1$: Um sistema onde se roda de líder a cada bloco, de modo a emular um algoritmo *Leader-Speaks-Once* (LSO) como o HotStuff [3].
(b) $k = 50$: Um sistema onde se roda de líder a cada 50 blocos.
(c) $k = 100$: Um sistema onde se roda de líder a cada 100 blocos.
(d) $k = \infty$: Um sistema onde não se altera a configuração. (Líder Estável)

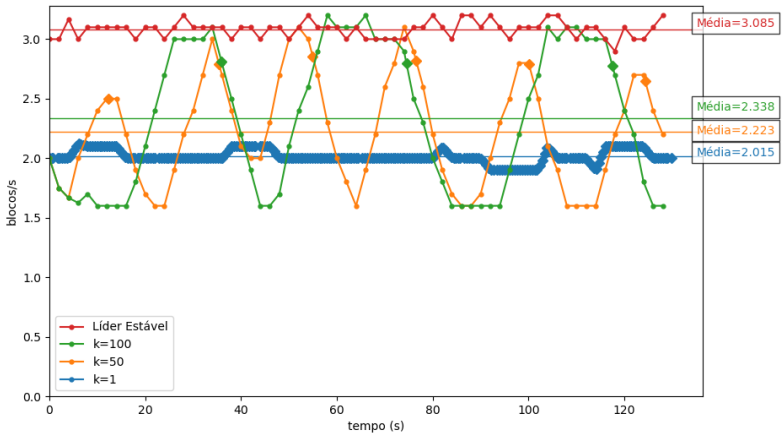

Figura 4: Débito (em blocos/s) ao longo do tempo entre quatro sistemas com frequências de rotação diferentes.

Os resultados são apresentados na Figura 4. Como era esperado, uma frequência de reconfiguração menor acaba por ter um débito maior. Isto acontece porque cada reconfiguração acarreta uma perda temporária no débito. Se o sistema reconfigurar frequentemente, o sistema não tem tempo para retomar o desempenho de um sistema estável antes de ocorrer uma nova reconfiguração. Estes resultados

sugerem que ao usar árvores de disseminação/ agregação o parâmetro k controla a tensão entre a otimização do débito e a mitigação de censura, sendo que a escolha do valor para k depende do risco/ impacto da censura e dos objetivos de qualidade de serviço definidos para o sistema.

4.4 Discussão

Os resultados mostram que a reconfiguração dinâmica, apesar de afetar negativamente o débito do sistema, é exequível em sistemas baseados em árvore. Em particular, os resultados apresentados na Figura 2 mostram que, mesmo com reconfiguração, um sistema em árvore tem um desempenho mais eficiente que um sistema em estrela, quando o tamanho do sistema esgota os recursos do líder. Ou seja, mesmo nos casos em que ter um líder estável não é desejável, os sistemas baseados em árvores de disseminação/ agregação conseguem oferecer melhorias no desempenho. Na concretização atual, uma das principais causas para a quebra do débito durante a reconfiguração deve-se à necessidade dos nós esperarem pelos blocos anteriores antes de participarem na nova configuração. Neste momento estamos a procurar maneiras de mitigar este efeito.

5 Conclusão e Trabalho Futuro

Neste artigo descrevemos uma solução que viabiliza complementar o Kauri com uma política de líder rotativo, implementado de forma a reduzir os custos inerentes a reconfiguração de árvores e com a possibilidade de tirar vantagem de um agendamento inteligente de árvores adaptadas para executarem em redes geo-distribuídas. Com base nos resultados preliminares da nossa avaliação experimental, partimos do ponto que algoritmos de consenso baseado em árvores não só são compatíveis com uma política de líder rotativo mas também possibilitam casos de uso interessantes a partir do mesmo. Como trabalho futuro, há duas vertentes de interesse em explorar: primeiramente, poderemos explorar a fundo uma avaliação ao sistema em vários casos diferentes de redes heterogéneas, com relevância em definir uma métrica para avaliar o desempenho esperado das árvores. A partir deste, a comparação de transições entre árvores diferentes de desempenhos semelhantes revelará otimizações para agendamentos preferíveis. Em segundo lugar, pretendemos no futuro permitir a alteração dinâmica dos agendamentos, de forma a responder a falhas e ou a alterações nas condições da rede. Estamos a trabalhar num sistema de monitorização e controlo tolerante a faltas Bizantinas para este efeito.

Agradecimentos. Este trabalho foi suportado pela Fundação para a Ciência e Tecnologia (FCT) por fundos nacionais através dos projectos INESC-ID UIDB/50021/2020, DACOMICO (via OE com ref. PTDC/CCI-COM/2156/2021), Ainur (via OE com ref. PTDC/CCI-COM/4485/2021) e ScalableCosmosConsensus.

Referências

[1] Marko Vukolić. "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication". In: *Open Problems in Network Security: IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*. Springer. 2016, pp. 112–125.

[2] Miguel Castro and Barbara Liskov. "Practical byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186.

[3] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. "HotStuff: BFT consensus with linearity and responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 2019, pp. 347–356.

[4] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation". In: *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 2021, pp. 35–48.

[5] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. "Enhancing bitcoin security and performance with strong consistency via collective signing". In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 279–296.

[6] Wenyu Li, Chenglin Feng, Lei Zhang, Hao Xu, Bin Cao, and Muhammad Ali Imran. "A scalable multi-layer PBFT consensus for blockchain". In: *IEEE Transactions on Parallel and Distributed Systems* 32.5 (2020), pp. 1146–1160.

[7] Eleftherios Kokoris-Kogias. "Robust and scalable consensus for sharded distributed ledgers". In: *Cryptology ePrint Archive* (2019).

[8] Helena Teixeira, Luus Rodrigues, and Miguel Matos. "Arvores de Disseminaçao e Agregaçao Cientes da Topologia para Suportar Consenso Bizantino em Larga Escala". In: *INForum* (2023).

[9] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. "Kollaps: decentralized and dynamic topology emulation". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20. Heraklion, Greece: Association for Computing Machinery, 2020.