Árvores de Disseminação e Agregação Cientes da Topologia para Suportar Consenso Bizantino em Larga Escala

Helena Teixeira, Luís Rodrigues e Miguel Matos

INESC-ID, Instituto Superior Técnico, U. Lisboa
{helena.teixeira,ler,miguel.marques.matos}@tecnico.ulisboa.pt

Resumo A utilização de árvores de disseminação e agregação permite suportar a execução de algoritmos de consenso bizantino em redes de larga-escala com elevado número de participantes. O trabalho anterior ou é agnóstico à topologia, ou pressupõe ambientes homogéneos. Neste trabalho propomos heurísticas cientes da topologia para criar árvores de disseminação e agregação em ambientes heterogéneos. Através de simulações, avaliamos o desempenho das nossas heurísticas em cenários realistas, mostrando que podem chegar a reduzir em 60% o tempo médio necessário para a recolha de um quórum.

1 Introdução

A capacidade de suportar a execução de algoritmos de consenso bizantino em redes de larga-escala com elevado número de participantes é um fator determinante para o desenvolvimento de blockchains altamente descentralizadas. Infelizmente, os algoritmos de consenso bizantino tradicionais [1,2] não possuem capacidade de escalar, uma vez que pelo menos um dos participantes necessita de trocar informação diretamente com todos os outros participantes.

Recentemente [3], a utilização de árvores de disseminação e agregação mostrou ser uma estratégia robusta para executar o consenso bizantino entre centenas de participantes. No entanto, o trabalho anterior baseado em árvores, ou é agnóstico à topologia da rede [1,2] ou pressupõe ambientes homogéneos [3]. O sistema Kauri, em particular, cria árvores aleatórias que são alheias à topologia e heterogeneidade da rede.

Mais detalhadamente, no Kauri é necessário criar uma sequência de árvores, em que os nós interiores numa das árvores são folhas nas restantes, de modo a suportar a reconfiguração no caso de falhas. Isto é conseguido dividindo os nós em vários grupos, em que cada grupo possui um número de nós suficiente para ocupar as posições interiores da árvore. O Kauri pressupõe, portanto, a existência de dois algoritmos: uma para decidir como dividir os nós em grupos, e outro para criar uma árvore a partir de um dado grupo, sendo que atualmente ambos os algoritmos são baseados em escolha aleatória.

Neste trabalho, pretendemos enriquecer o Kauri com heurísticas que têm em conta a topologia, em particular a latência, para criar árvores de disseminação e agregação cientes da topologia. Para tal, propomos heurísticas cientes da topologia para concretizar cada uma destas tarefas acima.

A nossa avaliação, baseada em simulações realísticas nas quais os participantes no consenso residem em centros de dados geograficamente distribuídos, mostra que as heurísticas propostas permitem ganhos de desempenho na ordem dos 60% relativamente à estratégia puramente aleatória usada pelo Kauri.

2 Trabalho Relacionado

O problema do consenso tem sido amplamente estudados pela comunidade de computação distribuída pelo que uma análise detalhada das diferentes alternativas está fora do âmbito deste trabalho. Dado o foco da nossa contribuição, a discussão que se segue centra-se no impacto da utilização de redes geo-distribuídas e heterogéneas na execução destes algoritmos.

2.1 Padrões de Comunicação

Muitos algoritmos de consenso Bizantino usam um padrão de comunicação *Todos-para-Todos* (como, por exemplo, o PBFT [1]) ou um padrão de comunicação *Um-para-Todos* (como, por exemplo, o Hotstuff [2]). Em ambos os casos, existe pelo menos um processo que tem de enviar (e receber) informação para (de) todos os outros processos o que cria não só pontos de estrangulamento no sistema mas também dificulta a paralelização da disseminação.

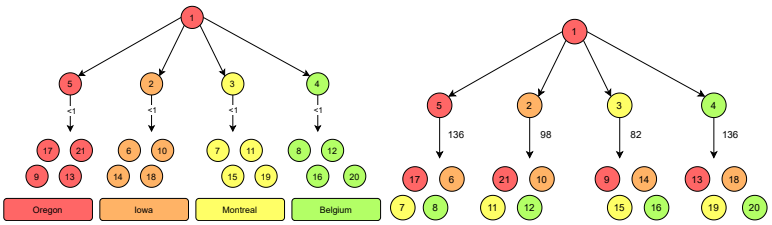
Para melhor distribuir a carga, algoritmos como o Kauri [3] usam um padrão de comunicação em árvore que serve simultaneamente para disseminar e agregar informação e onde a raiz da árvore é o líder do consenso. Quando um processo quer disseminar informação, envia esta informação para os seus filhos. Durante a agregação, um nó agrega a informação dos seus filhos e reencaminha para o pai. Este processo é repetido até chegar à raiz. Em termos de eficiência, o uso de árvores de disseminação e agregação representa um acréscimo de latência em comparação com sistemas como Hotstuff [2] que é compensado pelo uso de técnicas de encadeamento. Assim, tanto quanto apuramos os algoritmos de consenso bizantino existentes não têm a topologia em conta.

2.2 Os Desafios da Utilização de uma Rede Geo-Distribuída

Sistemas como o ResilientDB [4] mostraram que para obter bom desempenho em redes geo-distribuídas é crucial evitar cruzar ligações de elevada latência (como, por exemplo, entre diferentes centros de dados) múltiplas vezes e permitir que a propagação (ou agregação) de informação em zonas geograficamente distintas ocorra em paralelo. Note-se que a latência entre máquinas que se encontram no mesmo centro de dados ou região geográfica é significativamente menor

	Latência (ms)					
	O	I	M	B	T	S
Oregon (O)	$\leqslant 1$	38	65	136	118	161
Iowa (I)		$\leqslant 1$	33	98	153	172
Montreal (M)			$\leqslant 1$	82	186	202
Bélgica (B)				$\leqslant 1$	252	272
Taiwan (T)					$\leqslant 1$	137
Sidnei (S)						$\leqslant 1$

Tabela 1: Latência da comunicação entre diferentes centros de dados da Google medidos pelo ResilientDB [4].



(a) Configuração informada pela disposição geográfica dos nós.

(b) Configuração aleatória.

Figura 1: Comparação entre uma configuração informada pela disposição geográfica dos participantes e uma configuração aleatória. Os valores junto às setas indicam a latência máxima associada a essa ligação.

do que entre máquinas em regiões distantes, como se ilustra na Tabela 1. Isto requer que o algoritmo tenha em consideração a topologia da rede.

O padrão de comunicação em árvore permite paralelizar a disseminação de informação e, caso a configuração seja ciente, ou informada, das características da rede, pode favorecer a comunicação entre nós geograficamente próximos. A Figura 1 ilustra o impacto que uma configuração informada pode ter no tempo de disseminação. Consideremos 21 nós praticamente uniformemente distribuídos pelos 4 primeiros centros de dados da Tabela 1. Uma configuração ciente da rede, como a ilustrada na Figura 1a, favorece a comunicação local, pois o segundo nível da árvore não realiza nenhuma comunicação entre diferentes centros de dados o que permite reduzir substancialmente a latência da disseminação/ agregação. Por outro lado, uma configuração aleatória como a ilustrada na Figura 1b, não tem esta preocupação forçando a comunicação entre diferentes centros de dados nos dois níveis da árvore.

2.3 Robustez

Protocolos que usam um padrão de comunicação baseados em árvores de disseminação e agregação são mais vulneráveis a faltas que algoritmos que usam padrões *Todos-para-Todos* ou *Um-para-Todos*. Mais concretamente, enquanto nestes últimos o algoritmo consegue terminar desde que a rede esteja estável e o líder seja correto, num algoritmo baseado em árvore, a terminação só é garantida se todos os nós interiores da árvore forem corretos. Esta característica pode ser descrita de forma precisa pela definição de *árvore robusta*:

Definição 1 (Árvore robusta [3]) *Diz-se que uma aresta é segura se os vértices correspondentes são ambos processos corretos. Uma árvore é robusta se o processo líder estiver correto e, para cada par de processos corretos* (p_i) *e* (p_j)*, o caminho que liga estes processos é composto exclusivamente por arestas seguras.*

É pois fundamental que o processo de reconfiguração do sistema permita encontrar uma árvore robusta, pelo menos quando o número de faltas é pequeno, caso contrário a terminação pode não ser garantida. Caso não seja possível encontrar uma árvore robusta após um número pré-definido de reconfigurações, o sistema pode ser obrigado a adotar um padrão de comunicação menos eficiente, mas mais robusto como o padrão *Um-para-Todos*.

O processo de reconfiguração do Kauri [3] usa a seguinte estratégia para encontrar árvores robustas quando o número de faltas é pequeno. Os processos são divididos em t *grupos* disjuntos, cada um de tamanho maior ou igual a I, onde I é o número de nós internos numa árvore. Em seguida, são criadas árvores cujos nós internos são retirados exclusivamente de um determinado *grupo*, ou seja, cada árvore (T_i) é construída escolhendo um *grupo* (B_i), seguindo uma estratégia rotativa, e atribuindo nós do *grupo* (B_i) a todos os nós internos da árvore (incluindo a raiz).

É de notar que esta abordagem especifica quais os processos que devem ser internos (isto é todos os que pertencem ao grupo B_i selecionado), mas omite como se devem organizar esses nós entre si, bem como os restantes nós da árvore, sendo a árvore criada de forma aleatória. Como ilustrado no exemplo acima, isto leva a um desempenho sub-óptimo em redes heterogéneas. Na secção seguinte apresentamos a nossa proposta para combater este problema.

3 Construção de Árvores Cientes da Topologia

Nesta secção discutimos a nossa proposta para construir árvores cientes da topologia.

Modelo e Pressupostos. Como modelo de sistema assumimos os mesmos pressupostos do Kauri, e que são o padrão deste tipo de sistemas [3,1,2]. Em particular, assumimos um sistema composto por N processos, f dos quais podem ser bizantinos sujeito à restrição $f < N/3$. Os processos bizantinos podem comportar-se

arbitrariamente, mas não possuem poder computacional suficiente para subverter as primitivas criptográficas. Adicionalmente, assumimos a existência de um sistema capaz de fornecer as distâncias, em termos de latência, entre os diferentes processos do sistema. Na prática, isto pode ser concretizado com sistemas de coordenadas tais como o Newton [5]. No resto desta secção assumimos que o administrador do sistema define o número de processo N, o *fanout* da árvore, e o fator δ detalhado abaixo.

3.1 Abordagem

O nosso principal objetivo é desenvolver uma heurística para construir árvores de disseminação capazes de tirar partido da heterogeneidade da rede. A nossa abordagem parte de duas observações: i) a Internet apresenta características de uma rede *small-world* [6] e ii) os nós que participam em sistemas de blockchain com permissões estão tipicamente aglomerados em centros de dados geograficamente dispersos [4] que possuem internamente latências muito baixas e, entre eles, latências substancialmente mais elevadas tal como ilustrado na Tabela 1. Deste modo, a intuição da nossa abordagem passa por, no início da disseminação espalhar a informação por zonas geograficamente distantes, e, portanto, usando potencialmente ligações com maior latência, e daí em diante tirar partido da localidade e fazer o resto da disseminação usando ligações locais que tipicamente possuem latências mais baixas. Para atingir este objetivo precisamos de resolver dois sub-problemas: i) a alocação de nós a grupos e ii) a construção de uma árvore a partir de um grupo.

3.2 Alocação de Nós a Grupos

Dado o Kauri ter como ponto de partida um conjunto fixo de grupos, o primeiro desafio consiste em determinar como alocar os nós a grupos. Para tal, começamos por introduzir o conceito de δ-Aglomeração:

Definição 2 (δ-Aglomeração) *Uma δ-Aglomeração é um conjunto de nós que distam entre si, no máximo, δ unidades de latência.*

Partindo da informação fornecida pelo sistema de coordenadas geográficas, e do δ especificado pelo administrador, utilizamos um algoritmo de clustering [7] para dividir os nós em vários δ-aglomerados. Na prática, isto corresponde sensivelmente à divisão dos nós pelos centros de dados ou proximidade geográfica em que estes se encontram. De seguida, criamos t grupos inicialmente vazios, que correspondem aos grupos utilizados pelo Kauri para definir os nós internos da árvore (linhas 1– 3 do Algoritmo 1). Partindo dos δ-aglomerados previamente construídos, distribuímos os nós de cada δ-aglomerado pelos grupos B_i de uma forma rotativa tal como ilustrado na Figura 2 e definido nas linhas 4– 7 do Algoritmo 1. Deste modo, ao espalhar os nós de cada δ-aglomerado pelos diferentes grupos, maximizamos a diversidade de cada grupo B_i. Este facto é explorado no passo seguinte de construção de uma árvore a partir de um grupo.

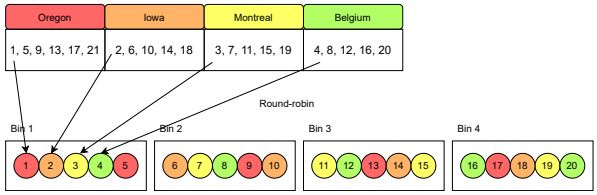


Figura 2: Distribuição de grupos baseado na distribuição geográfica.

Algoritmo 1: Alocação de nós a grupos

```
1 for i = 1, 2, ... (N/I) do
2 |   B_i ← ∅;
3 end for
4 for i = 1, 2, ..., I do
5 |   n ← nó de δ-aglomerado ainda não alocado a um grupo ;
6 |   B_i ← B_i ∪ {n} ;                    /* Grupo i com I nós criado */
7 end for
```

3.3 Construção de uma Árvore a Partir de um Grupo

A construção de uma árvore dado um grupo B_i implica uma sequência de passos, que detalhamos em seguida: i) seleção da raiz; ii) seleção da configuração dos restantes nós internos; iii) alocação das folhas aos nós internos. Esta lógica está definida mais formalmente no Algoritmo 2.

Seleção da raiz. A seleção da raiz é fundamental para o bom desempenho do sistema, pois a raiz atua como líder que, por sua vez, é responsável por iniciar cada uma das fases do algoritmo. Uma abordagem possível seria, para cada nó do grupo testar todas as combinações possíveis de árvores considerando esse nó como raiz. Esta abordagem, apesar de resultar na árvore ideal, não é prática dado que o número de combinações a explorar é exponencial. Deste modo, e seguindo o princípio enunciado acima de, no início da disseminação, isto é ao nível da raiz da árvore, espalhar a informação por zonas geograficamente distantes usamos a seguinte heurística: a raiz será um dos nós do grupo que pertence ao δ-aglomerado com menor latência média para os outros δ-aglomerados (linhas 1– 2 no Algoritmo 2). Em caso de empate, deve ser usada uma estratégia determinista como escolher o δ-aglomerado com o nó de identificador menor entre os empatados. A intuição é que a raiz deve comunicar com todas as zonas para iniciar o processo de disseminação, minimizando a latência necessária para esta disseminação global. Assim, queremos garantir que todos as zonas iniciam o processo de disseminação local a partir do primeiro nível, com o menor custo possível. A nível computacional, é linear em relação ao número de δ-aglomerados.

Configuração dos nós internos. Selecionada a raiz, o próximo passo é definir a configuração dos restantes nós do grupo B_i que serão os nós internos da árvore.

Algoritmo 2: Construção de árvores informadas

Input: Uma lista B_i com o grupo com I nós internos. M - aridade da árvore. Uma lista *restantes* com as folhas da árvore.
Output: Árvore T representada como lista.

```
1  ; Adiciona a raiz à árvore
2  raiz ← nó de B_i com menor latência média para os outros δ-aglomerados;
3  T ← T ∪ {raiz}
4  ; Disseminação global no primeiro nível da árvore
5  for internal = 1, 2, ... M do
6  |  T ← T ∪ {nó de um δ-aglomerado ainda não escolhido do grupo B_i}
7  end for
8  ; Número de níveis
9  L ← round(log_M(N + 1)) − 1
10 ; Caso exista mais que um nível de internos
11 parents ← nós mais profundos em T while L! = 1 do
12 |  for parent ∈ parents do
13 |  |  Ordena nós internos restantes de B_i por latência ao parent;
14 |  |  T ← M nós com menor latência;
15 |  end for
16 |  L ← L − 1;
17 end while
18 ; Aloca as folhas aos nós internos mais profundos
19 parents ← nós mais profundos em T
20 for parent ∈ parents do
21 |  Ordena nós de restantes por latência ao parent;
22 |  T ← M nós com menor latência;
23 end for
24 return T
```

É de notar que não só a escolha da raiz é importante, como descrito acima, mas também a ordem pela qual os nós internos aparecem na árvore é importante para obter um bom desempenho. Em concreto, a disseminação no Kauri acontece, por convenção, da esquerda para a direita, e portanto, os nós mais à esquerda deverão ser aqueles que têm uma ligação de menor latência para a raiz (linhas 5– 7 no Algoritmo 2). Isto é relevante, pois o algoritmo de consenso não precisa de esperar por todos os nós para fazer progresso, bastando um quórum. Deste modo, ao colocar os nós com menos latência para a raiz mais à esquerda, maximizamos a probabilidade de não ter que esperar pela resposta dos nós mais lentos que estão mais à direita. Caso a árvore tenha mais que dois níveis de nós internos (isto é a raiz e os filhos diretos) então comutamos a abordagem para priorizar a comunicação local. Neste caso, os nós internos são atribuídos a pais que pertencem ao mesmo δ-aglomerado ou, caso tal não seja possível, ao pai com menor latência (linhas 11– 17 no Algoritmo 2).

Alocação das folhas. Definida a configuração dos nós internos, resta alocar os nós dos restantes grupos como folhas da árvore. Neste ponto queremos maximizar a

comunicação local e, portanto, alocamos os nós folha de cada grupo a um nó pai que pertença ao mesmo δ-aglomerado ou, quando tal não é possível ao pai que minimiza a latência (linhas 20– 23 no Algoritmo 2). A nível computacional, estes últimos são lineares em relação ao número de nós internos e nós, respetivamente.

4 Avaliação

A avaliação experimental foi conduzida utilizando um simulador de eventos discretos em Python já usado noutros trabalhos [8,9]. O modelo de rede simulado segue a matriz de latências descrita na Tabela 1 sendo os processos distribuídos uniformemente por cada um dos seis centros de dados.

Codificámos os algoritmos descritos na Secção 3 no simulador e considerámos três cenários típicos: um cenário em que o *fanout* definido pelo administrador coincide com número de δ-aglomerados, um cenário em que o *fanout* é maior e outro em que é menor. Como principal métrica utilizamos o tempo necessário para a raiz recolher o quórum, que é o instante a partir do qual o algoritmo de consenso pode fazer progresso.

Para avaliar independentemente o impacto da heurística de alocação dos nós a grupos (Secção 3.2) e da construção de árvores a partir de um grupo (Secção 3.3) avaliamos, para cada cenário, as seguintes combinações: i) grupos informados com árvores informadas; ii) grupos informados com árvores aleatórias; iii) grupos aleatórios com árvores informadas; iv) grupos aleatórios com árvores aleatórias.

4.1 Cenário 1: M = número de δ-aglomerados

Para este cenário, tendo 6 δ-aglomerados, definimos o *fanout* $M = 6$ e criamos uma árvore regular de dois níveis. Para este efeito temos $N = 43$ nós. Para os grupos aleatórios geramos 10 grupos diferentes e para as árvores aleatórias geramos 100 árvores diferentes para cada grupo. Os resultados para este cenário nas quatro combinações estão apresentados na Figura 3.

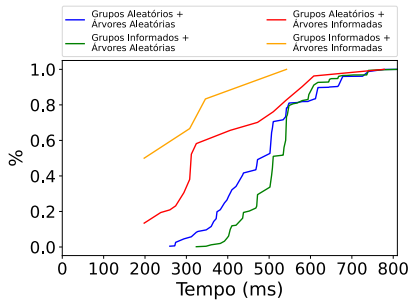
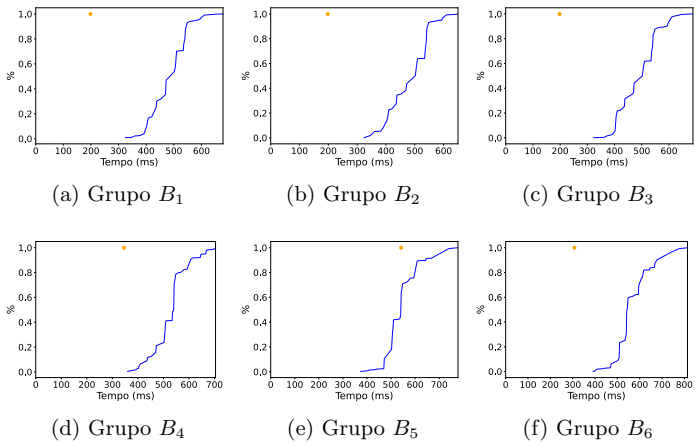


Figura 3: Tempo de recolha no Cenário 1.



(a) Grupo B_1 (b) Grupo B_2 (c) Grupo B_3

(d) Grupo B_4 (e) Grupo B_5 (f) Grupo B_6

Figura 4: Heurística informada (amarelo) vs estratégia aleatória (azul) no Cenário 1.

Começando pela construção de árvores aleatórias (linhas verde e azul), é possível verificar que oferecem um pior desempenho do que as soluções informadas. As diferenças entre construir árvores aleatórias a partir de uma distribuição por grupos aleatória ou informada não é significativa e, no caso desta figura, a diferença entre os resultados deve-se a não termos conseguido fazer uma amostra suficientemente grande. De facto, como se verá adiante, estas linhas aparecem cada vez mais sobrepostas nos restantes cenários. Por sua vez, o algoritmo informado para construir árvores já apresenta ganhos significativos, mesmo quando a constituição dos grupos continua a ser aleatória; neste caso observamos um aumento do desempenho em 40% em metade das árvores. A distribuição informada dos nós pelos grupos melhora ainda mais os resultados (na ordem dos 60%), pois aumenta a probabilidade de existir, em todos os grupos, uma raiz bem situada.

A Figura 4 mostra o impacto da combinação das diferentes heurísticas. Em particular, selecionámos uma distribuição dos nós por grupos e comparamos a árvore informada que é construída pela nossa heurística a partir de cada grupo (ponto amarelo), com várias árvores aleatórias que podem ser construídas a partir do mesmo grupo (linha azul). Como se pode observar, para a maioria dos grupos, a árvore informada apresenta melhor desempenho do que qualquer das 100 árvores construídas aleatoriamente, o que demonstra que a probabilidade de obter uma boa árvore de forma não informada é geralmente baixa. No entanto, a figura mostra também que a nossa heurística nem sempre consegue obter a melhor árvore. Na Figura 4e, por exemplo, o grupo B_5 inclui 2 nós em Taiwan e 1 nó em cada um dos restantes δ-aglomerados. Apesar de o δ-aglomerado com menor latência média para os outros δ-aglomerados neste cenário ser em Iowa, é possível encontrarmos uma árvore mais eficiente com a raiz em Taiwan.

4.2 Cenário 2: $M >$ número de δ-aglomerados

Mantendo os 6 δ-aglomerados usados anteriormente, definimos o *fanout* $M = 10$ e criámos uma árvore regular de dois níveis. Para este efeito usamos $N = 111$ nós. Os resultados para este cenário estão apresentados nas Figuras 5 (resultados agregados para várias distribuições de nós por grupos) e na Figura 6 (para uma distribuição em particular). Como se pode verificar, as várias soluções apresentam o mesmo padrão relativo, embora neste caso as vantagens das árvores informadas sejam mais evidentes. Isto deve-se ao facto de, ao termos um *fanout* maior, temos um fator de paralelismo maior na disseminação, que é melhor explorado pelas árvores informadas.

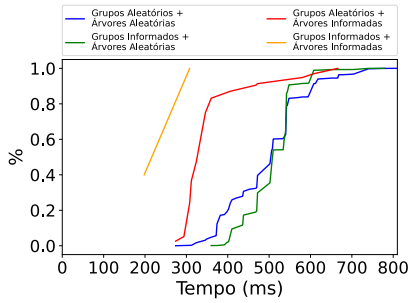
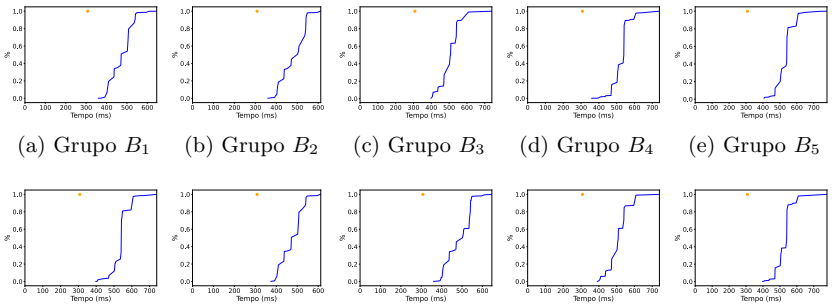


Figura 5: Tempo de recolha no Cenário 2.



(a) Grupo B_1 (b) Grupo B_2 (c) Grupo B_3 (d) Grupo B_4 (e) Grupo B_5

(f) Grupo B_6 (g) Grupo B_7 (h) Grupo B_8 (i) Grupo B_9 (j) Grupo B_{10}

Figura 6: Heurística informada (laranja) vs estratégia aleatória (azul) no Cenário 2.

4.3 Cenário 3: $M <$ número de δ-aglomerados

Por fim, avaliamos um cenário em que o *fanout* é menor que o número de δ-aglomerados o que tipicamente implica aumentar a profundidade da árvore. Para tal, mantemos os 6 δ-aglomerados, definimos o *fanout* $M = 3$ e criamos uma árvore regular de 3 níveis. Para este efeito usámos $N = 40$ nós. Os resultados para este cenário estão apresentados na Figura 7 e na Figura 8, usando uma metodologia idêntica à usada nos cenários anteriores. Mais uma vez, fica claro que usar estratégias informadas permite obter tempos de recolha menores. No entanto, neste cenário, a diferença entre usar uma distribuição dos nós pelos grupos informada ou aleatória não é significativa. Isto explica-se em parte pelas mesmas razões do cenário anterior - ao ter um *fanout* menor as árvores informadas conseguem explorar menos o fator de paralelismo o que dilui a diferença entre grupos informados e aleatórios. É de notar contudo que o aumento da profundidade tem um impacto também nos resultados que requer um estudo mais aprofundado.

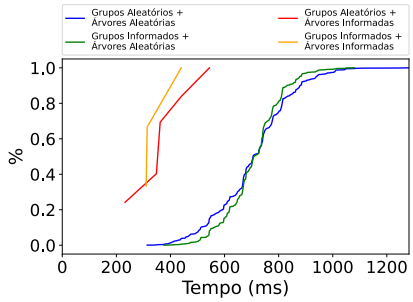
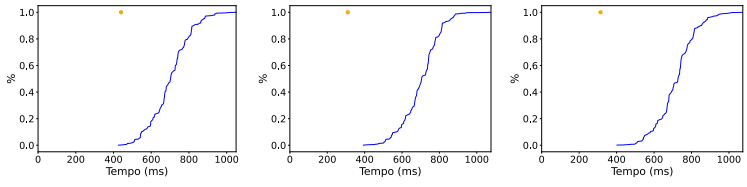


Figura 7: Tempo de recolha (Cenário 3).



(a) Grupo B_1 (b) Grupo B_2 (c) Grupo B_3

Figura 8: Heurística informada (laranja) vs estratégia aleatória (azul) no Cenário 3

4.4 Discussão

A avaliação considera um conjunto relativamente pequeno de cenários mas que cobre diferentes relações entre o número de δ-aglomerados e o *fanout* da árvore, isto é, cenários em que o *fanout* é igual (Cenário 1), superior (Cenário 2) ou inferior (Cenário 3) ao número de δ-aglomerados. Em qualquer dos cenários, a nossa heurística de construção e árvores consegue, em média, tempos de recolha muito inferiores às árvores aleatórias atualmente utilizadas pelo Kauri. Para além disso, em todos os casos, a divisão informada dos nós pelos grupos consegue melhorar os resultados da heurística usada para construir uma árvore de modo informado. Este efeito é menos pronunciado no Cenário 3, mas seriam precisas mais experiências para perceber se é o valor do *fanout* ou a profundidade da árvore o fator que mais afeta o impacto da distribuição informada de nós por grupos. A avaliação mostra também que existem agrupamentos para os quais é fácil encontrar, mesmo de forma aleatória, árvores com melhor desempenho do que aquelas encontradas pela nossa heurística (por exemplo, o caso apresentado na Figura 4e). Estamos a investigar se é possível melhorar a heurística para evitar estes casos sem a complicar desnecessariamente (uma vantagem da atual versão é que a sua execução é muito eficiente do ponto de vista computacional).

5 Conclusão e Trabalho Futuro

Neste trabalho apresentamos uma solução que visa enriquecer o Kauri com heurísticas para a geração de árvores de disseminação e agregação baseadas na topologia da rede. Neste contexto, propomos uma heurística para a construção das árvores que usa informação sobre a latência entre os nós para criar os agrupamentos e outra para gerar árvores a partir destes agrupamentos. Avaliamos ambas as heurísticas recorrendo a simulações baseadas em latências de rede realistas e apresentamos resultados que sugerem ser possível reduzir o tempo necessário para recolher um quórum Bizantino na ordem dos 60%. Como trabalho futuro pretendemos avaliar a nossa abordagem num cenário com código real a executar em centros de dados geo-distribuídos. Neste trabalho, consideramos também que os parâmetros M e δ são fornecidos pelo administrador do sistema; seria interessante estender o trabalho para configurar também estes parâmetros de forma automática.

Agradecimentos: Este trabalho foi suportado pela FCT – Fundação para a Ciência e a Tecnologia, através dos projectos UIDB/50021/2020, DACOMICO (PTDC/CCI-COM/2156/2021) e Ainur (PTDC/CCI-COM/4485/2021), e desenvolvido no âmbito do projeto no. 51 "BLOCKCHAIN.PT - Agenda Descentralizar Portugal com Blockchain", financiado por Fundos Europeus, nomeadamente "Plano de Recuperação e Resiliência - Componente 5: Agendas Mobilizadoras para a Inovação Empresarial", incluído no programa de financiamento NextGenerationEU.

Referências

1. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OsDI. Volume 99. (1999) 173–186
2. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. (2019) 347–356
3. Neiheiser, R., Matos, M., Rodrigues, L.: Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. (2021) 35–48
4. Gupta, S., Rahnama, S., Hellings, J., Sadoghi, M.: Resilientdb: Global scale resilient blockchain fabric. arXiv preprint arXiv:2002.00160 (2020)
5. Seibert, J., Becker, S., Nita-Rotaru, C., State, R.: Newton: Securing virtual coordinates by enforcing physical laws. IEEE/ACM Transactions on Networking **22**(3) (2014) 798–811
6. Albert, R., Jeong, H., Barabási, A.L.: Diameter of the world-wide web. nature **401**(6749) (1999) 130–131
7. Schaeffer, S.E.: Graph clustering. Computer science review **1**(1) (2007) 27–64
8. Matos, M., Felber, P., Oliveira, R., Pereira, J.O., Rivière, E.: Scaling up publish/subscribe overlays using interest correlation for link sharing. IEEE Transactions on Parallel and Distributed Systems **24**(12) (2013) 2462–2471
9. Matos, M., Mercier, H., Felber, P., Oliveira, R., Pereira, J.: Epto: An epidemic total order algorithm for large-scale distributed systems. In: Proceedings of the 16th Annual Middleware Conference. Middleware '15, New York, NY, USA, Association for Computing Machinery (2015) 100–111