P-KAD: Enriquecer o Kademlia com Particionamento

João Monteiro[1], Pedro Ákos Costa[1], Alfonso de la Rocha[2], Yiannis Psaras[2] e João Leitão[1]

[1]NOVA LINCS & DI/FCT/NOVA
[2]Protocol Labs

Resumo Tabelas de dispersão distribuídas (DHT, do inglês *distributed hash table*), permitem a criação de sistemas descentralizados de armazenamento e procura de ficheiros capazes de superar as limitações de abordagens centralizadas (escalabilidade, tolerância a falhas, custos operacionais, entre outras). Estes sistemas normalmente distribuem tanto os participantes como o conteúdo publicado de forma uniforme pelo espaço de identificadores. Desta forma, é possível que pedidos a conteúdo próximo (e.g., num contexto geográfico ou aplicacional) tenham de transitar por participantes distantes até chegar ao destino. Em sistemas cujo padrão de acesso seja maioritariamente a conteúdo próximo, esta abordagem pode limitar a escalabilidade e o tempo de resposta do sistema.
Neste artigo, propomos dois novos esquemas de organização topológica tendo como base a rede estruturada Kademlia, que permitem reduzir o caminho percorrido por pesquisas na rede, assumindo que estas têm elevada localidade geográfica. Para este efeito, recorremos a prefixos adicionados aos identificadores dos participantes que codificam a sua área geográfica. O primeiro esquema - partição suave - utiliza identificadores prefixados em que todos os participantes pertencem a uma única DHT, cuja topologia fica enviesada para promover uma maioria de ligações entre pares próximos, garantindo que participantes geograficamente próximos são também próximos no espaço de identificadores da DHT. O segundo esquema - partição rígida - explora uma alternativa onde os participantes se juntam a diferentes DHTs dependendo do seu prefixo, recorrendo a um serviço externo para efetuar pesquisas em zonas diferentes. Ambos os esquemas foram avaliados utilizando uma rede emulada com 2000 participantes, e mostram uma redução na latência média na resolução de pesquisas no sistema quando comparados com o desenho original, o qual é mais notória quando uma porção significativa das pesquisas são por conteúdos localizados na mesma região geográfica (i.e., por participantes com o mesmo prefixo).

1 Introdução

Redes sobrepostas estruturadas [11,9,13] possibilitam a criação de múltiplas categorias de aplicações e sistemas [7,4,2]. Em particular, as redes sobrepostas estruturadas estão na base das tabelas de dispersão distribuídas (DHTs) normalmente

utilizadas como base de sistemas descentralizados de armazenamento e procura de ficheiros [2]. O interesse em soluções descentralizadas está a aumentar com o aparecimento da Web 3.0 [1,2,12] que se foca em soluções descentralizadas com o controle do lado dos utilizadores finais. Exemplos desta nova geração, são sistemas baseados em *blockchain* [12,14], e os produtos da empresa Protocol Labs[1], o IPFS [2] e FileCoin [3] que utilizam uma DHT como base para os seus serviços.

Neste tipo de sistemas, identificadores únicos, uniformemente distribuídos, são atribuídos tanto aos participantes como aos recursos na rede de forma a ser possível localiza-los, sendo responsável pelo conteúdo o participante cujo identificador esteja mais próximo do identificador do recurso. Nestas redes, tipicamente cada participante mantém tabelas de encaminhamento locais, prioritizando pares cujo identificador esteja mais próximo do seu.

Infelizmente, conforme o sistema cresce, a atribuição de identificadores pode criar redes onde participantes conhecem uma maioria de pares bastante distantes considerando a rede física, causando assim um aumento de latência geral no sistema e uma limitação à escalabilidade do mesmo [8,10]. Este desafio é maior quando o padrão de acesso a recursos é maioritariamente a conteúdo gerado em localizações próximas (i.e., num contexto geográfico), pois pode ser necessário transitar por participantes distantes com latências altas até o conteúdo ser localizado.

Para endereçar este desafio, neste artigo propomos dois novos esquemas de organização topológica de rede, tendo como base a rede estruturada Kademlia [9], com o principal objetivo de otimizar o caminho percorrido por pesquisas de recursos com alta localidade geográfica. Para este efeito, recorremos à modificação dos identificadores dos participantes através da adição de prefixos que codificam a sua área geográfica (ou região). O primeiro esquema, utiliza um particionamento suave, em que utilizando a codificação de prefixos, é possível enviesar a rede para que participantes da mesma região estejam mais próximos na rede sobreposta, promovendo uma maior conetividade entre estes. O segundo esquema utiliza um particionamento rígido em que participantes se juntam a diferentes DHTs dependendo dos seus prefixos. De forma a transitar entre DHTs para realizar pesquisas fora da região local é utilizado um serviço externo ao protocolo da DHT, que mantém pontos de presença em cada região (i.e., DHT). Note-se que apesar de neste artigo darmos ênfase a localização geográfica, os prefixos podem também codificar uma aplicação que executa acima da DHT (no caso da DHT suportar a operação de múltiplas aplicações), extraindo assim possíveis benefícios para nós que executam a mesma aplicação, sendo que nós da mesma aplicação iram frequentemente aceder a conteúdo da aplicação que executam, tornando assim as pesquisas maioritariamente locais (a uma parte da DHT).

Este documento está organizado da seguinte forma: Na Secção 2 apresentamos trabalho relacionado. Na Secção 3 apresentamos com detalhe a nossa proposta de arquitetura. Na Secção 4 discutimos os resultados experimentais obtidos e finalmente, concluímos o artigo na Secção 5.

[1] https://protocol.ai

2 Trabalho Relacionado

Tabelas de dispersão distribuídas são um tipo de redes sobrepostas estruturadas [9,11,13]. Estas estruturas operam sobre um espaço virtual de identificadores que são atribuídos, de forma aleatória, aos participantes e aos recursos (i.e., conteúdos) do sistema, mapeando os mesmos para um ponto na rede. Cada participante utiliza uma tabela de encaminhamento para escolher os pares onde registam e procuram conteúdos, sendo que esta é preenchida e mantida de forma diferente dependendo do protocolo. Por exemplo, no Chord [11] os participantes formam uma topologia de anel sendo que cada entrada i da tabela de endereçamento de um participante com um identificador de m bits, é preenchida pelo par com uma distância de $2^{i+1} mod\ 2^m$ saltos entre eles.

Outras topologias utilizam métricas de distância entre participantes de modo a otimizar o preenchimento destas tabelas. O Kelips [6], coloca os participantes em grupos escolhidos através do identificador criado. Um participante preenche a sua tabela de encaminhamento com pares tendo como base medições de latência tanto para dentro do grupo como para grupos exteriores. O Coral [5], cria vários grupos hierárquicos, com base em medições de latências, utilizando outras DHT (e.g., Kademlia [9], Chord [11]). Nesta topologia, um participante junta-se ao grupo com menor latência média entre ele e todos os participantes, criando assim uma visão da rede que dá preferência a pares com menores latências. Infelizmente, estes protocolos operam de forma oportunista não dando garantias de consistentemente reduzir a latência de pedidos com alta localidade.

Para encontrar recursos e outros participantes na rede, são utilizados protocolos em que as mensagens são encaminhadas para zonas progressivamente mais próximas do identificador alvo até chegar ao destino pretendido. No Tapestry [13], quando um participante recebe uma mensagem que está a ser encaminhada para um identificador destino, caso a mensagem não seja para si, encaminha-a para um participante cujo identificador deve ter mais um dígito em comum com o destino em relação ao nó local.

As soluções propostas neste artigo são baseadas na DHT Kademlia [9]. Esta DHT foi escolhida devido ao modo como a tabela de encaminhamento (i.e., utilizando uma métrica entre identificadores de XOR) é preenchida, uma vez que permite que a prefixação dos identificadores tenha o impacto que pretendemos na topologia de rede. Adicionalmente, esta é a DHT base do sistema IPFS [2], o qual motiva parcialmente este trabalho. Note-se que as soluções propostas neste artigo podem ser aplicadas a outras DHTs.

O Kademlia [9] é um dos protocolos de DHT mais populares sendo utilizado em múltiplas aplicações [2,4,12]. O Kademlia opera sobre um espaço de identificadores com m bits, utilizando uma métrica de XOR para calcular distâncias entre identificadores. A métrica calcula a operação binária de XOR entre dois identificadores obtendo um número inteiro que codifica a semelhança usada como distância entre os identificadores.

A tabela de encaminhamento de cada participante é constituída por múltiplas listas, uma para cada $i \in [0, m[$, chamadas de *k-buckets*, com um tamanho máximo de k, compostas por pares cuja operação de XOR entre os seus identifi-

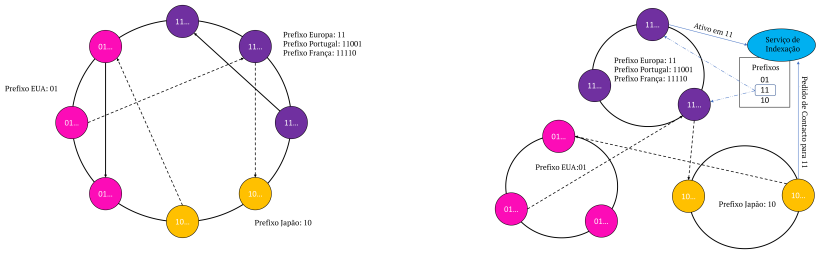
cadores e o do participante local pertence ao intervalo de $[2^i, 2^{i+1}]$. Inicialmente, cada participante só possui uma única lista que cobre todo o espaço de identificadores e à medida que novos pares vão sendo conhecidos, é preenchida. Caso uma lista esteja cheia, um procedimento de divisão é despoletado se o identificador do próprio participante estiver contido no intervalo de identificadores que a lista suporta. Desta forma, um dado participante conhece uma maioria de pares próximos do seu identificador.

Para possibilitar a comunicação entre participantes, a DHT Kademlia suporta quatro chamadas de procedimentos remotos (RPC, do inglês *Remote Procedure Call*). O primeiro, e mais importante, é o FindNode em que dado um identificador como chave, localiza o participante responsável por essa chave. Para isto, o participante inicia um processo recursivo em que contacta k dos seus pares conhecidos na tabela de endereçamento que estejam mais próximos da chave. Estes respondem com mais k participantes que conhecem que estejam próximos da chave, oferecendo ao iniciante, pares que estejam ainda mais próximos da chave que ele de seguida consegue contactar e continuar o processo de pesquisa. O processo acaba quando não existem pares mais próximos da chave que ainda não foram contactados.

O procedimento FindValue é utilizado para encontrar recursos guardados por outros participantes da rede. O funcionamento do procedimento é semelhante ao anterior, sendo que termina quando recebe o recurso representado pela chave em vez de k participantes. O procedimento Store é utilizado para guardar recursos na rede. Este procedimento é iniciado com a criação de um identificador para o recurso sendo de seguida utilizado o FindNode para encontrar os participantes mais próximos do identificador. Finalmente, o recurso é guardado nos k participantes mais próximos resultados da pesquisa. Os recursos guardados têm um tempo de vida útil, necessitando o publicador original de realizar pedidos de Store periodicamente para manter o recurso válido. Por fim, o procedimento Ping é utilizado para verificar se o participante ainda está ativo.

Um participante ao entrar na rede, executa um FindNode procurando pelo seu próprio identificador, começando pelo seu contacto inicial, de modo a criar uma primeira vista da rede e dos seus vizinhos mais próximos, utilizada para iniciar a sua tabela de encaminhamento local. Periodicamente cada participante executa esta operação de modo a atualizar a sua visão das suas imediações na rede.

Quando é realizado um pedido de FindValue, o participante cria de seguida pedidos de Store para os k pares mais próximos do identificador que não enviaram uma resposta com o conteúdo durante a operação do FindValue. Isto permite propagar conteúdos mais popular por mais participantes de forma a lidar com pontos quentes na rede (i.e., zonas no espaço de identificadores com um grande número de pedidos), distribuindo os conteúdos por mais participantes na rede.



(a) Particionamento Suave. (b) Particionamento Rígido.

Figura 1: Representação visual dos dois esquemas.

3 Solução Proposta

De modo a promover a localidade na DHT, a cada participante que se junta ao sistema, é adicionado ao seu identificador o prefixo que codifica a sua área geográfica, sendo esta obtida através do endereço IP e de um serviço externo. Os prefixos são estaticamente definidos e disponíveis para acesso a todos os participantes do sistema. É possível estabelecer uma hierarquia entre prefixos em que os bits mais significativos do prefixo codificam uma região mais abrangente e os menos significativos aumentam a especificidade (e.g., continente, país, etc.).

Ao efetuar esta modificação aos identificadores, é possível otimizar a procura de pares e conteúdo com proximidade geográfica devido ao funcionamento da tabela de endereçamento da DHT Kademlia, pois um participante terá conhecimento de mais participantes cujo prefixo seja o mesmo ou próximo. Para isso, o conteúdo publicado por um participante, é também ele prefixado com os mesmos bits desse participante. Note-se que à semelhança do sistema IPFS [2], assumimos que os clientes obtêm a chave de um recurso por um processo *out-of-band*.

A prefixação dos identificadores, cria efetivamente, partições no espaço de identificadores da rede sendo que neste artigo exploramos dois métodos de particionamento da rede: o particionamento suave onde todos os participantes pertencem a uma única DHT; o particionamento rígido onde os participantes se juntam a diferentes DHTs dependendo do seu prefixo, recorrendo a um serviço externo para efetuar pesquisas nas restantes DHTs. Na Figura 1 estão representados os dois esquemas propostos, simplificando a topologia do Kademlia a uma forma de anel.

3.1 Partição Suave

O particionamento suave, provoca a criação de múltiplas partições virtuais no espaço de identificadores da DHT, ao adicionar aos identificadores o prefixo que codifica a região geográfica tanto dos participantes como do conteúdo publicado. Ao aplicar este esquema ao funcionamento base do Kademlia, a tabela de endereçamento de um dado participante terá uma tendência para ser preenchida por pares cujo prefixo seja igual ao próximo. Esta alteração tem como efeito

principal um decréscimo do número de saltos necessários para comunicar com participantes na mesma zona, não sendo necessário contactar participantes em regiões longínquas.

Para armazenar conteúdo, o participante que inicia o protocolo, adiciona ao identificador do conteúdo o seu prefixo de modo a marcar a região de origem do conteúdo, sendo o restante procedimento efetuado de maneira semelhante ao protocolo base. Sendo o conteúdo prefixado, este será armazenado em pares com o prefixo igual ou próximo, desta forma colocando o conteúdo numa região da rede que irá originar a maioria dos pedidos a este conteúdo assumindo elevada localidade nos acessos a recursos (previamente observado pela equipa do IPFS). Para outros participantes localizarem e obterem conteúdo na rede, é necessário conhecer à *priori* tanto o prefixo como o identificador do conteúdo.

3.2 Partição Rígida

O particionamento rígido induz a criação de múltiplas partições disjuntas, formadas a partir dos vários prefixos (ou pequenos conjuntos de prefixos consecutivos); cada uma representando uma DHT. Devido a esta separação e ao diâmetro mais curto de cada rede, pesquisas locais (i.e., dentro da partição) apresentam um menor número médio de saltos.

Para executar o procedimento de pesquisa em partições remotas, os participantes dependem de um serviço externo, a que chamamos de *serviço de indexação*, e que atua fora do protocolo da DHT. O serviço de indexação contém pontos de acesso (i.e., outros participantes) para as outras regiões servindo como uma ponte entre partições. O serviço de indexação pode operar de forma centralizada, onde uma única instância conhece pontos de acesso; ou descentralizada em que diferentes instâncias conheçem pontos de acesso diferentes, sem afetar a correção do protocolo de DHT.

Um participante, ao juntar-se à rede, cria uma ligação a pelo menos uma das instâncias do serviço de indexação, sendo a informação de contacto passada no processo de inicialização. De forma a atualizar a informação no serviço de indexação com pontos de acesso ativos, os participantes enviam periodicamente uma notificação de como estão ativos. Os procedimentos de pesquisa na rede (i.e., FindNode e FindValue) apenas necessitam de acesso ao serviço quando os pedidos requerem comunicação com participantes com um prefixo diferente, sendo o pedido realizado de forma semelhante ao procedimento base caso o prefixo seja o mesmo.

De forma a minimizar o número de pedidos efetuados ao serviço de indexação e assim reduzir a latência de pedidos a conteúdos remotos, cada participante mantém uma *cache* com os últimos pontos de acesso a uma dada região (i.e., prefixo) que contactou. Assim, quando é realizada uma pesquisa fora da DHT local, um participante verifica se tem pontos de acesso para a DHT remota, e caso existam executa a pesquisa através desses pares. Caso contrário, o participante é obrigado a contactar o serviço de indexação para obter pontos de acesso à DHT remota, guardando os mesmos na sua cache local para a realização de pesquisas futuras para a mesma DHT remota. Os pares são removidos da cache de um

participante se estes não responderam a um pedido, ou se caso o seu tempo de vida na cache tenha expirado.

No procedimento de STORE, o conteúdo é armazenado noutros pares da mesma partição apenas, pois é esperado que a maioria dos pedidos de acesso sejam realizados por participantes da partição local. Caso um dado conteúdo se torne popular, é possível replicar o conteúdo em várias partições através da republicação por parte de participantes dessas partições.

3.3 Discussão

As nossas soluções pretendem minimizar o tempo de resposta de pedidos em padrões de acesso com alta localidade. Comparando as duas soluções propostas, a partição suave, sendo uma abordagem mais simples, tem um menor impacto no funcionamento do sistema como um todo. No entanto, apesar da sua simplicidade, o impacto que cria na organização topológica da rede sobreposta é uma grande vantagem em relação à versão original quando o padrão de acessos é dominado por pedidos a conteúdos publicados na mesma região. O particionamento rígido, para além da diminuição da latência para pedidos na mesma região, tem a vantagem adicional de criar pontes entre regiões distantes utilizando o serviço de indexação, diminuindo o número de saltos necessários para as contactar.

4 Avaliação

Nesta secção descrevemos o ambiente de avaliação experimental, seguido da discussão dos resultados obtidos relativamente às duas soluções propostas.

4.1 Ambiente Experimental

De forma a avaliar o desempenho das nossas soluções propostas comparando com o Kademlia sem alterações, recorremos à emulação de uma rede sintética com 2000 participantes. A rede foi gerada com a ajuda de uma biblioteca *python*[2] da seguinte forma: 2000 nós são colocados num plano 2D, em que cada eixo representa um intervalo [0,1] e criam-se ligações entre todos os nós que se encontrem a uma distância menor do que 0.15 unidades. De modo a tornar a rede mais realista, em que um nó tem muitas ligações próximas e algumas ligações para outros nós mais distantes, iteramos sobre todas as ligações criadas e trocamos uma ligação próxima por uma ligação distante com uma probabilidade de 0.001 (note-se que o grafo tem um total de 32.980 ligações). De forma a criarmos configurações de rede que representassem diferentes zonas geográficas no grafo gerado calculámos 3, 5 e 10 partições, recorrendo a um modelo estocástico de blocos[3]. A Figura 2 representa dois dos modelos criados em que as várias cores

[2] https://graph-tool.skewed.de

[3] O modelo estocástico de blocos calcula uma número configurável de comunidades, ou blocos, no grafo tento em conta os de arcos existentes entre os vértices

representam as várias partições. Para obter a latência entre todos os participantes, calculámos o caminho mais curto entre participantes, resultando numa matriz com os valores dos caminhos. Para emular latências que façam sentido numa implementação real do sistema, multiplicámos a matriz por 1000 unidades (codificando 1 segundo), obtendo uma matriz com latência média aproximada de 1 segundo.

Nas experiências realizadas, utilizámos a plataforma com máquinas de alto desempenho *Grid 5000*[4]. Utilizamos 4 máquinas cada uma com 18 *cores* e 96GB de RAM. Executamos protótipos de ambas as soluções propostas e do protocolo Kademlia como comparação, os quais foram desenvolvidos em Go. Estes protótipos foram executados em contentores Docker, em que cada contentor corre vários processos que utilizam o protocolo UDP para comunicar entre si, como especificado no artigo original do Kademlia. Para aplicar as latências anteriormente descritas, cada contentor Docker ao iniciar, aplica regras da ferramenta Linux `tc`, que aplicam tempos de espera ao envio de cada pacote de cada processo que corre no contentor para todos os outros processos do sistema.

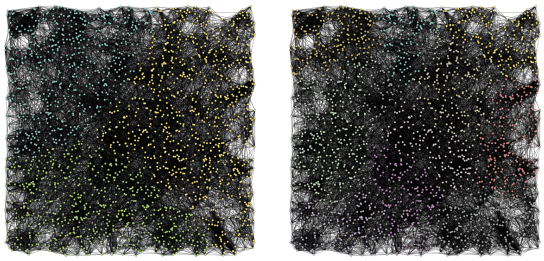
Para comparar as nossas soluções com o Kademlia original, focámos a avaliação na medição da latência de um pedido (FindNode e FindValue). A DHT, tem essencialmente 3 parâmetros que alteram o comportamento dos pedidos: k é o parâmetro que regula o número de participantes guardados em cada um dos *k-buckets* como mencionado na Secção 2. O parâmetro k também controla o número de contactos enviados como resposta no pedido de FindNode e o número de contactos utilizados em cada ronda de pedidos para todos os RPC. O parâmetro α regula o número de pedidos enviados em paralelo para todos os RPC. O parâmetro β estabelece o número de respostas necessárias para progredir para a próxima ronda de um pedido utilizando os contactos recebidos. Nas experiências realizadas, os parâmetros k, α e β foram configurados com os valores, respetivamente, de 5, 3 e 2. Assume-se, nestas experiências, que perdas de mensagens são eventos raros.

Relativamente ao serviço de indexação, necessário à solução baseada em particionamento rígido, o número de participantes armazenados para cada uma das partições é 5 sendo que estes são atualizados periodicamente, a cada 20 segundos. Nas experiências existe apenas uma instância deste serviço.

Para testar o desempenho das nossas soluções com perfis de pedidos diferentes, variámos a probabilidade de cada pedido ser realizado a um conteúdo publicado por um participante localizado na região local entre 1%, 25%, 50%, 75%, 90% e 95%. Cada experiência tem um tempo de execução de 20 minutos sendo excluídos os primeiros 5 minutos para a rede estabilizar. A cada 5 segundos, cada participante tenta procurar por um conteúdo, sendo medido o tempo e número de saltos até ser encontrado o conteúdo solicitado.

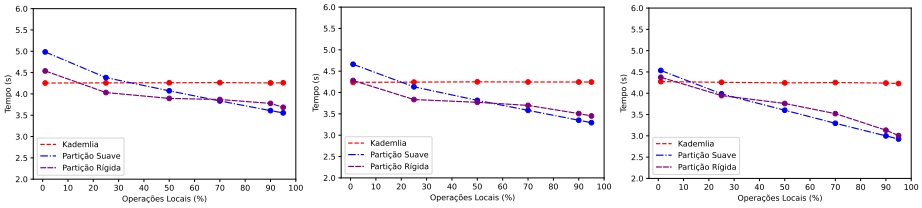
Nas experiências relativas ao FindValue, cada participante armazena na rede 20 conteúdos diferentes. Um participante, para armazenar o conteúdo na rede, guarda o conteúdo localmente e envia como valor o seu endereço. Após

[4] A platforma Grid5000 é suportada pelo grupo de investigação alojado no Inria e inclui CNRS, RENATER e outras Universidades e organizações (ver www.grid5000.fr).



(a) Grafo com 3 partições. (b) Grafo com 10 partições.

Figura 2: Exemplo dos grafos de participantes criados.



(a) 3 partições. (b) 5 partições. (c) 10 partições.

Figura 3: Latência de resposta de FindNode.

o armazenamento, são executados pedidos pelas chaves guardadas, seguindo as distribuições mencionadas anteriormente. O pedido é satisfeito quando o participante que realiza a pesquisa vai buscar o conteúdo ao publicador original. Para não ser criada uma sobre-replicação na rede, causada pelo grande número de pedidos num curto espaço de tempo, quando um participante pede um conteúdo, este não é replicado na receção. No entanto, como está especificado no protocolo original, cada participante irá armazenar a chave e o valor obtido (i.e., o endereço do nó que contém o conteúdo), nos k participantes mais próximos da chave que não conheciam o valor.

4.2 Resultados

Nesta secção, analisamos os resultados das experiências relativas ao tempo de resposta do FindNode e do FindValue, comparando as várias soluções (Partição Suave e Rígida) com o Kademlia sem alterações.

As Figuras 3a, 3b e 3c apresentam os resultados para pedidos com 3, 5 e 10 partições respetivamente, representando no eixo do x a percentagem de operações locais e no eixo do y a latência média em segundos de todos os pedidos. Nestas figuras, podemos observar que em ambas as nossas soluções ao aumentar o número de partições, os tempos médios de resposta naturalmente

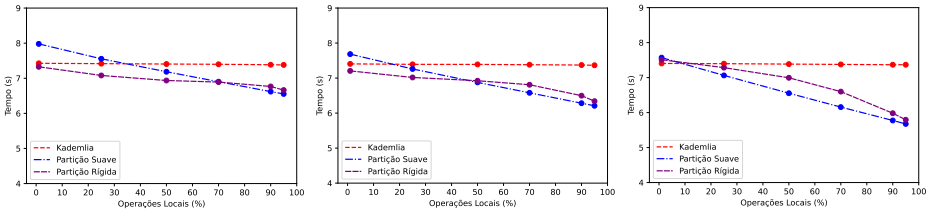
diminuem devido aos participantes que fazem parte de cada partição terem uma menor distância média entre eles. O Kademlia original, ao aumentar a localidade dos pedidos, não vê uma diminuição nos seus tempos de resposta devido a cada participante preencher a sua tabela de endereçamento com participantes aleatórios. Em cada uma das figuras, à medida que se aumenta a percentagem de pedidos locais é possível observar uma diminuição considerável nos tempos de resposta para as duas soluções, confirmando que o enviesamento da tabela de endereçamento no caso do particionamento suave, e a separação dos participantes em várias partições disjuntas no particionamento rígido, têm o efeito pretendido.

É possível observar o efeito que a cache tem no particionamento rígido nos resultados entre o 1% e os 25% de pedidos locais. Para uma percentagem de pedidos locais de 1%, no particionamento rígido, cada participante, devido a manter a cache sempre válida, só realiza um pedido ao serviço de indexação e consequentemente, existe a possibilidade de bastantes participantes possuírem os mesmos pontos de acesso para partições externas, tendo a consequência do desempenho degradar ligeiramente devido à sobrecarga de pontos de acesso. No entanto, para maiores percentagens de pedidos locais é possível ver o efeito positivo da cache com o decréscimo acentuado na latência. A consequência da sobrecarga de pontos de acesso é menos visível nas nossas experiências com mais partições onde existe uma maior diversidade de escolha de partições remotas.

O particionamento suave, sofre também uma descida de desempenho no caso de um número elevado de pedidos a partições exteriores devido ao número de pares para estas ser reduzido e, tendo todos os participantes o mesmo ponto de contacto na sua inicialização, as suas tabelas de endereçamento são enviesadas pela visão deste participante causando o mesmo efeito que no particionamento rígido. No entanto, o particionamento suave, à medida que o número de partições aumenta, também vê o seu desempenho a melhorar para estes casos (com 1% pedidos locais). Isto deve-se a uma melhor distribuição dos identificadores pelo espaço, permitindo assim um aumento da probabilidade de um identificador de um participante (especialmente fora da partição local) se encontre num dos *k-buckets* e consiga ser adicionado à tabela de encaminhamento, criando uma maior diversidade nas conexões estabelecidas.

Na Figura 3c, para os pedidos entre 50% e 90%, o particionamento rígido não obtém uma latência média de resposta tão baixa como o do particionamento suave, devido a mensagens destinadas a partições externas necessitarem ocasionalmente de pedidos ao serviço de indexação, aumentando ligeiramente o tempo de comunicação.

As Figuras 4a, 4b e 4c representam a latência média de resposta para a operação FindValue nos mesmos cenários. Nestas figuras podemos constatar uma maior acentuação das diferenças entre as soluções propostas e o Kademlia original. De novo, à medida que o número de pedidos locais vai aumentando, o desempenho das nossas soluções aumenta. Estes resultados devem-se à maior proximidade entre participantes que não necessitam de utilizar conexões com grande latência para encontrar as chaves pesquisadas, acentuados pela neces-



(a) 3 partições (b) 5 partições (c) 10 partições

Figura 4: Latência de resposta de FindValue

sidade da mensagem adicional para pedir o conteúdo ao endereço encontrado. Nestas tipo de operação, o efeito observado para o FindNode, no caso do particionamento rígido para uma percentagem baixa de pedidos locais já não é visível, visto que a necessidade de obter o conteúdo pesquisado desacelera o ritmo de emissão de pedidos evitando assim sobrecarregar os nós que servem de pontos de contacto em partições remotas.

4.3 Discussão

Comparando o particionamento suave com a DHT original, o benefício de adicionar prefixos aos identificadores é visível para pedidos com grande localidade. O sacrifício da pequena degradação de desempenho para pedidos externos parece ser razoável tendo em conta o observado de que muitos pedidos no IPFS [2] são para conteúdos publicados na mesma região geográfica, onde os nossos benefícios de desempenho são muito notáveis.

Por sua vez, o particionamento rígido, consegue, nos casos em que existe um equilíbrio entre a utilização da cache e o número de pedidos efetuados a conteúdo publicado em regiões remotas, ter um melhor desempenho que a DHT original nas duas categorias de pedidos (i.e.,FindNode e FindValue). É importante notar que a escolha do tempo de vida da cache nos participantes é um fator que pode ter impacto no desempenho assim como o número de instâncias do serviço de indexação e o número de participantes guardados para cada partição (não só devido a churn, mas também por questões de distribuição de carga). Explorar mecanismos para otimizar a operação e desempenho do serviço de indexação será abordado em trabalho futuro.

5 Conclusão

Neste artigo apresentámos duas propostas alternativas de solução capazes de melhorar o desempenho da DHT Kademlia quando o padrão de acesso a recursos na rede tem alta localidade. Exploramos uma alternativa, o particionamento suave, que estende o identificador da cada participante, ao adicionar um prefixo que codifica a sua região geográfica possibilitando ao protocolo original otimizar os

elementos contidos nos *k-buckets* de cada participante para promover localidade na rede sobreposta. A segunda alternativa explorada consiste em criar múltiplas DHT, uma para cada região (i.e., prefixo), em que os participantes utilizam um serviço de indexação com conhecimento de pontos de acesso em todas as regiões para aceder a estas. Esta última alternativa permite a criação de atalhos na rede onde participantes de regiões distantes conseguem comunicar num menor número de rondas.

Os resultados experimentais, retirados de um ambiente realista com 2000 participantes e várias partições, mostram que ambas as soluções conseguem ter um melhor desempenho que a DHT original, em termos de latência média, para cargas de trabalho com alta localidade sem sacrificar o desempenho de pedidos que não apresentem esta propriedade.

Referências

1. Web 3.0 technology stack. https://web3.foundation/about/, accessed July 2021
2. Benet, J.: IPFS - Content Addressed, Versioned, P2P File System. Tech. Rep. Draft 3 (2014), http://arxiv.org/abs/1407.3561
3. Benet, J., Greco, N.: Filecoin: A Decentralized Storage Network. Tech. rep. (2017), https://filecoin.io/filecoin.pdf
4. Cohen, B.: The bittorrent protocol specification, https://www.bittorrent.org/beps/bep_0003.html
5. Freedman, M.J., Mazieres, D.: Sloppy hashing and self-organizing clusters. In: International Workshop on Peer-to-Peer Systems. Springer (2003)
6. Gupta, I., Birman, K., Linga, P., Demers, A., Van Renesse, R.: Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In: International Workshop on Peer-to-Peer Systems. Springer (2003)
7. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: OceanStore: An architecture for global-scale persistent storage. SIGPLAN Notices (ACM Special Interest Group on Programming Languages) (2000)
8. Leitão, J.: Topology Management for Unstructured Overlay Networks. Phd thesis
9. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: International Workshop on Peer-to-Peer Systems. Springer (2002)
10. Ratnasamy, S., Stoica, I., Shenker, S.: Routing algorithms for dhts: Some open questions. In: International workshop on peer-to-peer systems. Springer (2002)
11. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Comput. Commun. Rev. **31**(4) (Aug 2001)
12. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Tech. rep. (2014), arXiv:1011.1669v3
13. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications **22**(1) (2004)
14. Zyskind, G., Nathan, O., Pentland, A.S.: Decentralizing privacy: Using blockchain to protect personal data. In: 2015 IEEE Security and Privacy Workshops (2015)