ResEst – Algoritmo Distribuído para a Inferência de Recursos da Rede *

Vítor Hugo Menino, Pedro Ákos Costa e João Leitão

NOVA LINCS & DI/FCT/NOVA University of Lisbon, Lisboa, Portugal

Resumo Nos sistemas distribuídos descentralizados onde os nós não têm uma vista global da rede, não é trivial um nó saber a sua capacidade relativa à rede em termos de recursos. Isto acontece por várias razões, tais como o facto de cada nó apenas conhecer uma fração reduzida de outros nós, para além do dinamismo existente nestes tipos de redes (entrada e saída de nós). No entanto, estimar o quão poderoso um nó é em relação aos outros nós da rede pode ser muito importante para diversas aplicações, por exemplo para realizar uma distribuição de carga adequada entre eles conforme os seus recursos em relação ao resto da rede, beneficiando o sistema como um todo.
Neste artigo apresentamos um algoritmo descentralizado cujo objetivo é estimar quão poderoso um nó é relativamente ao resto da rede, através da obtenção de um histograma que codifica a distribuição da capacidade de todos os nós existentes. O algoritmo permite ajustar a confiança pretendida nos resultados estimados de modo a permitir algum controle entre os recursos consumidos/rapidez do algoritmo e a qualidade da estimativa obtida. O algoritmo foi avaliado através de simulação, mostrando que com um intervalo de confiança de 95% para um erro menor do que 15%, um nó consegue obter uma estimativa dos recursos da rede em 54 passos de comunicação numa rede com 1.000.000 nós.

Keywords: Sistemas Descentralizados · Algoritmos Descentralizados · Inferência de Distribuição de Capacidade.

1 Introdução

Os sistemas descentralizados foram alvo de intensos esforços de investigação devido à sua promessa de escalabilidade [7,13,15,18] — visto os recursos destes sistemas aumentarem naturalmente com o aumento do número de participantes — e robustez — devido à remoção de pontos de falha únicos — para além de interesses na descentralização administrativa. A relevância destes sistemas está a aumentar novamente devido às recentes propostas no contexto da Web 3.0 [5,22,1] que aposta em arquiteturas e serviços de grande escala descentralizados.

* Este trabalho foi parcialmente financiado pela FC&T através dos proj. NOVA LINCS (UID/CEC/04516/2013) e do projeto NG-STORAGE (PTDC/CCI-INF/32038/2017).

Em inúmeras destas aplicações [22,5] é útil e potencialmente essencial que um nó (i.e., um processo/máquina que participa no sistema descentralizado suportando a operação de uma ou várias aplicações) consiga obter uma estimativa das capacidades dos restantes nós existentes na rede, de modo a saber onde se insere nessa escala. Um exemplo onde existe esta necessidade é em sistemas distribuídos onde se pretende realizar uma distribuição de carga adequada entre os vários nós conforme os seus recursos em relação ao resto da rede [6,22], permitindo a nós mais poderosos contribuírem mais, e evitando a sobrecarga de nós com menos recursos.

Em sistemas que recorrem a um coordenador central [23,10,16], um nó pode trivialmente obter uma estimativa das capacidades dos restantes nós da rede — visto que a entidade central conhece todos os nós e, consequentemente, pode facilmente saber quais os seus recursos, partilhando depois com todos os outros. O mesmo acontece em sistemas distribuídos descentralizados em que os nós têm vistas globais [13,20] (i.e., todos os nós conhecem os restantes), pois cada nó pode facilmente pedir a todos os outros que lhe enviem a sua capacidade e pode obter facilmente uma visão das capacidades dos outros nós na rede. Este tipo de soluções acarretam custos de manutenção muito elevados quando se considera sistemas de grande escala.

De facto, muitas das aplicações distribuídas de tamanho considerável que operam no mundo real [5,22] não usam vistas globais, pois, de um ponto de vista prático, o custo de comunicação (e potencialmente de manutenção) de modo a manter esta informação atualizada é demasiado elevado, não só devido ao grande número de nós existentes, mas também devida à alta frequência de entrada e saída destes [9] (i.e., *churn*). Por estas razões, muitas soluções recorrem a vistas parciais — onde cada nó conhece apenas uma fração da totalidade de nós do sistema, sendo que o número de nós conhecidos tipicamente evolui de forma logarítmica com a dimensão do sistema [14].

Porém, nestes sistemas (em que nós têm apenas vistas parciais da rede), não é trivial estimar (localmente) a distribuição dos recursos de todos os participantes da rede. Consequentemente, é difícil um nó identificar a relação entre os seus recursos disponíveis localmente e o resto da rede, por forma a ajustar localmente a sua contribuição para o sistema, evitando que nós menos poderosos fiquem sobrecarregados e que os nós mais poderosos possam contribuir mais, de forma a promover uma melhor operação do sistema como um todo [6] (potencialmente em troca da obtenção de benefícios, questão essa que é ortogonal à contribuição deste artigo).

Por conseguinte, neste artigo apresentamos um algoritmo descentralizado cujo objetivo é estimar quão poderoso um nó é relativamente ao resto da rede, através da obtenção de um histograma que aproxima a distribuição da capacidade de todos os nós existentes. Este algoritmo permite ajustar a confiança pretendida na estimativa computada de modo a permitir algum controle entre os recursos consumidos pelo algoritmo (assim como o seu tempo de execução) e a qualidade da estimativa obtida.

A estrutura do artigo é a que se segue: na Secção 2 apresentamos e discutimos o trabalho existente nesta área. Na Secção 3 apresentamos o algoritmo distribuído desenvolvido, que se chama *ResEst*. Na Secção 4 descrevemos a nossa configuração experimental, os resultados obtidos nas experiências elaboradas e discutimos os mesmos. Finalmente, na Secção 5, concluímos o artigo fazendo um resumo das contribuições e identificamos os próximos passos.

2 Trabalho Relacionado

Definir propriedades dos participantes constituintes de um sistema foi alvo de grande interesse, levando ao aparecimento de algoritmos de particionamento de rede [8,11] (i.e., *network slicing*) e de protocolos de população [2,20]. Os algoritmos de particionamento de rede têm como por objetivo criar partições de rede de forma a organizar os recursos de rede numa escala global, no entanto, este objetivo apesar de semelhante ao deste artigo, é fundamentalmente diferente, pois o desafio que endereçamos é permitir que um nó consiga decidir por si como melhor contribuir para a operação do sistema, identificando a relação da sua capacidade com o resto da rede. Por outro lado, os protocolos de população tentam fazer também uma caracterização do sistema, no entanto, estes sofrem da ausência de uma condição de paragem determinista.

Contudo, para um nó conseguir inferir as capacidades dos restantes nós da rede (para conseguir estimar onde se encontra nessa escala), é necessário usar algum tipo de agregação de dados na rede. Como tal, nesta secção analisamos vários protocolos de agregação existentes na literatura.

Um dos algoritmos de agregação mais relevante é o Extrema Propagation [3]. Este algoritmo funciona através de vetores gerados pelos nós e comunicação por disseminação por rumor. Apesar de ser passível de ser utilizado em sistemas de larga escala (devido ao erro da estimativa ser dependente dos tamanhos dos vetores trocados entre nós, que é parametrizável), este algoritmo apenas calcula uma estimativa da função de agregação soma, sendo que um nó não consegue inferir propriedades relativas ao seu contributo na rede.

Por outro lado, o Q-Digest [19] permite o cálculo de funções de agregação mais complexas, tais como a moda e a mediana. Neste algoritmo, cada nó computa uma estrutura de dados (designada *q-digest*). Estas estruturas de dados são propagadas através de uma árvore até um determinado nó coletor. Apesar das funções de agregação complexas que o algoritmo oferece, este está dependente de uma rede estruturada (uma árvore), o que o torna bastante frágil em redes de grande dimensão que são mais afetadas por *churn*.

Outro algoritmo de agregação relevante é o Randomized Reports [4], que funciona através de uma sondagem probabilística da rede, de modo a contar o número total de nós. Neste algoritmo cada um nó realiza o envio — através de disseminação por inundação — de uma mensagem para os restantes nós da rede. Quando um destes nós recebe a mensagem, responde ao remetente original com uma probabilidade parametrizável do algoritmo. O nó original calcula (probabilisticamente) quantos nós estão na rede sem todos estes terem respondido

à mensagem de sondagem. Ainda que este algoritmo tente evitar a sobrecarga da rede com grandes quantidades de mensagens ao minimizar a quantidade de nós que respondem ao remetente original, apresenta limitações na sua funcionalidade ao apenas conseguir calcular contagens da rede. Mesmo enriquecendo o algoritmo para capturar mais informação, o facto de recorrer à disseminação por inundação torna o algoritmo dispendioso do ponto de vista de comunicação.

Por outro lado, existem algoritmos de agregação que nos oferecem uma contagem de nós mais complexa e que são mais leves nos mecanismos de comunicação que utilizam. Um destes exemplos é o Random Tour [17]. Ainda que o objetivo deste algoritmo seja estimar o número de nós existentes na rede, este pode ser também usado (como referido pelos autores) para contar apenas os nós que satisfaçam um determinado critério (e.g., RAM total superior a 1024MB). Desta forma, este pode ser usado por um nó para inferir (de uma forma elementar) os recursos da rede.

O Random Tour funciona da seguinte forma. O nó que quer saber quantos nós existem na rede envia uma mensagem com um contador a um vizinho seu escolhido de forma aleatória. Este vizinho incrementa o contador da mensagem por $1/d$ (onde d representa o número de vizinhos do nó) e envia a mensagem para um vizinho seu escolhido de forma aleatória. Este processo repete-se até a mensagem voltar ao remetente original. Quando esta finalmente volta, o remetente original calcula uma estimativa do número de nós da rede, multiplicando o valor do contador pelo seu número de vizinhos.

Este algoritmo consegue, de uma forma simples, contar o número de nós na rede que satisfazem um determinado critério. No entanto, a sua operação não permite recolher informação suficiente para facilmente identificar qual deve ser o seu contributo para a operação do sistema como um todo. Para além disso, o Random Tour apresenta uma outra limitação crítica, que é o facto de, para a estimativa ser feita, as mensagens terem de retornar ao remetente original desta. No entanto, como as mensagens são trocadas entre nós de forma aleatória, se a rede for de alguma dimensão, a probabilidade de uma mensagem voltar ao seu originador é muito reduzida, tornando o algoritmo lento a obter estimativas. Na próxima secção, tiramos partido de intuições do Random Tour (mais especificamente, no seu padrão de comunicação) para construir o ResEst — um algoritmo distribuído para a inferência de recursos dos nós em redes com vistas parciais.

3 ResEst

A nossa solução — o *ResEst* — consiste em utilizar mensagens que seguem um padrão de comunicação por passeios aleatórios, em que cada mensagem constroi um histograma com as capacidades dos nós por onde ela passa. O cálculo destes histogramas é inspirado pela forma como o Random Tour faz contagens de nós com uma certa propriedade. Na nossa solução, por cada nó que processa uma mensagem, o nó verifica em que classe do histograma ele se identifica e incrementa o contador dessa mesma classe em uma unidade. Note-se que, no Random Tour, a incrementação do contador é feita por $1/d$ (em que d é o número de vi-

zinhos do nó), ao contrário da nossa solução, onde incrementamos o contador por 1. Isto deve-se ao facto de assumirmos que todos os nós da rede sobreposta sobre a qual o ResEst opera têm o mesmo número de vizinhos, o que pode ser garantido ao usar-se um protocolo de gestão de filiação como o HyParView [15].

Para efetivamente um nó conseguir calcular a estimativa, as mensagens iniciadas por ele têm de ser capazes de retornar a si mesmo. No entanto, para evitar incorrer em limitações de outras soluções, utilizamos um critério de paragem para a coleção de dados concreto, que leva a mensagem a ser retornada ao nó que a originou — *a margem de erro máxima* da média dos valores colecionados pela mensagem, para um intervalo de confiança [21] parametrizável.

O cálculo da margem de erro da média estimada para um determinado intervalo de confiança é realizado através da Fórmula 1. Esta computação depende de três variáveis: *i*) n — o número de amostras obtidas; *ii*) σ — o desvio padrão das amostras obtidas; e *iii*) z — o coeficiente de confiança para o intervalo de confiança desejado e para o número de amostras obtidas até esse momento.

$$marginErr = z * (\sigma/\sqrt{n}) \tag{1}$$

A intuição desta solução é que à medida que mais amostras se vão colecionando na mensagem, um desvio padrão elevado da média (que resultará em margens de erro maiores) das capacidades irá ser amortizado pelo número de amostras. Sendo que em sistemas em que as capacidades dos nós são uniformes, uma mensagem precisará de menos saltos para obter uma margem de erro mais baixa que o limite máximo parametrizado, isto porque o desvio padrão será pequeno. Por outro lado, um sistema que tenha nós com capacidades mais heterogéneas, a mensagem precisará de recolher mais amostras para amortizar o peso do desvio padrão elevado. Em suma, este critério de paragem não depende do tamanho do sistema, mas sim da qualidade da amostra.

Note-se que o algoritmo beneficia de uma distribuição uniforme das capacidades dos nós da rede. Para redes em que tal não seja o caso, de modo a aumentar a fiabilidade do algoritmo, as mensagens devem conseguir colecionar uma amostra da rede que seja representativa. Para aumentar a probabilidade de amostras de boa qualidade, a rede sobre a qual o algoritmo opera deve ser não estruturada com ligações entre nós criadas de forma aleatória, em que os nós têm um grau de entrada semelhante, distribuindo assim a probabilidade de um nó ser escolhido como alvo do passeio aleatório de forma uniforme. Uma exemplo de uma protocolo de rede sobreposta que dá estas propriedades é o HyParView [15].

No Algoritmo 1 apresentamos o pseudocódigo simplificado do ResEst, simplificando a componente do algoritmo que começa o processo do passeio aleatório. Note-se que este pode ser um processo periódico para recalcular as estimativas. No Algoritmo 1 apresentamos, no entanto, a parte fundamental do algoritmo que é o processamento de mensagens (Alg. 1 linha 1). Uma mensagem no nosso algoritmo contem a *média* dos valores de capacidade dos nós por onde esta passou (a média é calculada guardando a contagem dos nós e a soma das capacidades), o *histograma* contendo contagens de nós em várias classes de capacidades, e o nó que originou a mensagem (i.e., *originador*). O Algoritmo recebe dois parâmetros,

Algoritmo 1: ResEst

```
//Parâmetros
      margemErro //Margem de erro máxima
      confInt //Intervalo de confiança

1. Upon ReceiveMessage(m, originador) from emissor do:
2.     if MargemAceitavel(m) then:
3.       trigger send(m) to originator
4.     else:
5.       vizinho ⟵ EscolheVizinhoAleatorio()
6.       trigger send(m) to vizinho

7. Procedure MargemAceitavel(m):
8.     m.media ⟵ CalculaEAtualizaMedia(m)
9.     m.histograma ⟵ AtualizaHistograma(m)
10.    erro ⟵ CaculaMargemErro(m,confInt)
11.    taxaDeErro ⟵ erro/m.media
12.    return taxaDeErro ≤ margemErro
```

a $margemErro$ e o $confInt$, que respetivamente representam a margem de erro máxima e o intervalo de confiança a utilizar. Quando um nó processa uma mensagem (Alg. 1 linha 1), o nó calcula se a nova margem de erro após adicionar o seu contributo à mensagem é menor que a margem de erro máxima aceitável. Este procedimento (Alg. 1 linha 7) começa por atualizar a média presente na mensagem chamando a função CalculaEAtualizaMedia(m) (Alg. 1 linha 8) que aplica o valor de capacidade do nó local aos valores presentes na mensagem. De seguida, o histograma presente na mensagem é também ele atualizado (função AtualizaHistograma(m) no Alg. 1 linha 9). Por fim, a margem de erro dos valores presentes na mensagem é calculada, recorrendo à Fórmula 1 descrita anteriormente e codificada pela função CaculaMargemErro(m,$confInt$) no Alg. 1 linha 10, e uma taxa desse erro calculada (Alg. 1 linha 11) de forma a ser comparável com o parâmetro $margemErro$ (Alg. 1 linha 12) que codifica o erro máximo aceitável na amostra presente na mensagem. O resultado desta comparação é utilizado como critério de paragem do passeio aleatório feito pela mensagem. No caso da taxa de erro ser menor que a margem de erro máxima aceitável, a mensagem é enviada para o seu originador (Alg. 1 linha 3). Caso contrário, a mensagem continua a ser propagada para um vizinho aleatório do nó local (Alg. 1 linhas 5 e 6).

4 Resultados Experimentais

Esta secção está dividida em duas subsecções. Na primeira, descrevemos a configuração experimental utilizada para validar a nossa solução — o ResEst. Na segunda, apresentamos e analisamos os resultados obtidos.

4.1 Configuração Experimental

A nossa solução tem como objetivo operar sobre redes de grande dimensão (e.g., com um milhão de nós). Como tal, para validarmos a parte matemática do nosso

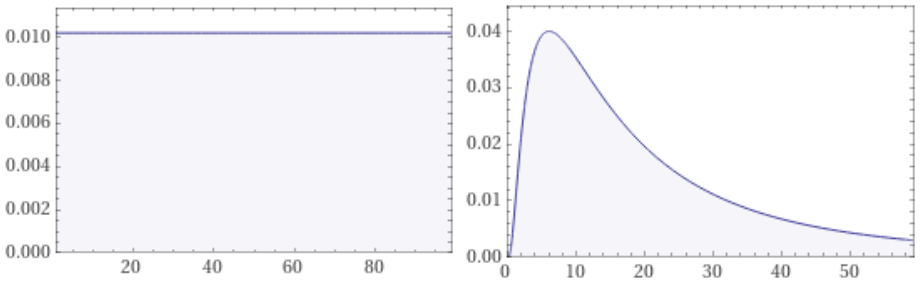
algoritmo, desenvolvemos um simulador[1]. Este simulador ignora as complexidades da rede (i.e., propriedades da rede, latência e largura banda entre nós, falhas de comunicação, entre outros), focando-se em executar o algoritmo descrito anteriormente. Neste simulador, e nas nossas experiências, cada nó apenas tenta obter uma estimativa dos recursos do sistema, ou seja, cada nó apenas origina um passeio aleatório.

Do ponto de vista de um nó, o simulador captura a movimentação da mensagem entre nós e calcula a margem de erro em cada passo (para um intervalo de confiança que é um parâmetro da experiência). Quando a margem de erro é menor que um parâmetro da experiência, o simulador apresenta o histograma calculado para essa mensagem terminado assim o fluxo de execução da mesma. A simulação termina quando todos os fluxos de execução de mensagens terminam. Em cada passo da simulação da comunicação de uma mensagem, o simulador simula a operação de um protocolo de amostragem de pares [12] em que escolhe aleatoriamente 7 ou 14 nós (respetivamente ao número de nós da experiência — 1.000 ou 1.0000.000 — o tamanho das vistas parciais é o logaritmo natural do número total de nós do sistema), entre todos os nós da experiência, obtendo assim uma vista parcial para o nó. De seguida, o simulador escolhe um alvo para a mensagem de forma aleatória da vista parcial do nó. Note-se que os nós têm um valor de capacidade que foi atribuído de acordo com um ficheiro de configuração que contém as capacidades de todos os nós. Nas nossas experiências usámos duas distribuições de capacidades diferentes, uma distribuição uniforme e uma log-normal. As capacidades dos nós para cada uma destas distribuições foram geradas uma única vez e guardadas em ficheiros de configuração diferentes.

Nas nossas simulações, utilizamos quatro parâmetros para controlar as experiências: *i*) *Margem de Erro Máxima*, que representa uma percentagem referente à taxa de erro máxima aceitável da média de capacidades para o passeio aleatório terminar (i.e., para ativar o critério de paragem); *ii*) *Intervalo de Confiança*, que consiste numa percentagem que indica a confiança existente no valor estimado da média sobre a amostra obtida; *iii*) *Distribuição*, que indica a distribuição das capacidades dos diferentes nós da rede, que é uma distribuição uniforme ou log-normal (i.e., que ficheiro de configuração utilizar); *iv*) *Número de nós*, que indica o número de nós que são simulados.

Nas nossas experiências variámos estes parâmetros entre os seguintes valores. Para a margem de erro máxima, utilizámos 25% e 15%. Para o intervalo de confiança utilizámos 90% e 95%. Para a distribuição, como referido, foram utilizadas uma distribuição uniforme e uma log-normal; a distribuição uniforme é controlada por dois valores — min e max — que limitam o espaço de valores possíveis, os valores utilizados para gerar a configuração de capacidades uniforme foram $min = 1$ e $max = 99$; a distribuição log-normal é controlada por dois valores — μ e σ — que respetivamente codificam o valor esperado (i.e., média) e o desvio padrão do logaritmo natural da variável, os valores utilizados para gerar a configuração de capacidades log-normal foram $\mu = 2,8$ e $\sigma = 1$. Na Figura 1a encontra-se a função de distribuição de probabilidade da distribuição

[1] https://github.com/hisetip/resest-sim



(a) Distribuição uniforme.

(b) Distribuição log-normal.

Figura 1: Funções de distribuição de probabilidade.

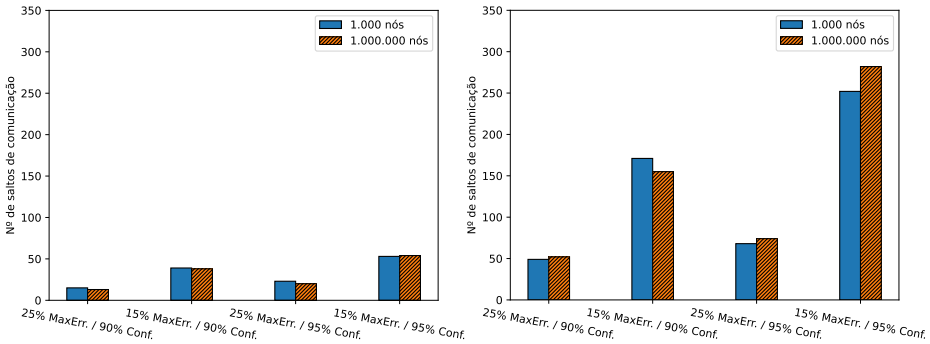
uniforme, onde a média das capacidades dos nós é 50, com o número de nós mais fracos do que a média sensivelmente igual ao número de nós mais poderosos do que esta. Na Figura 1b encontra-se a função de distribuição de probabilidade da distribuição log-normal, onde se representa uma rede mais realista. A seleção destes parâmetros específicos ($\mu = 2,8$ e $\sigma = 1$) foi realizada experimentalmente, de forma a simular uma rede onde existem muitos nós moderadamente fracos e alguns nós poderosos — à semelhança de cenários reais [6].

Por fim, o número de nós foi variado entre 1.000 e 1.000.000. As nossas experiências foram feitas sobre a combinatória dos parâmetros acima descritos (resultando em 16 experiências diferentes) e os seus resultados são apresentados nas Tabelas 1 e 2, para experiências com uma distribuição uniforme e log-normal para as capacidades dos nós da rede, respetivamente. Cada experiência diferente foi executada 10 vezes para remover possíveis ruídos experimentais. Os resultados mostram a média das 10 execuções. De seguida fazemos a análise dos resultados obtidos.

4.2 Resultados

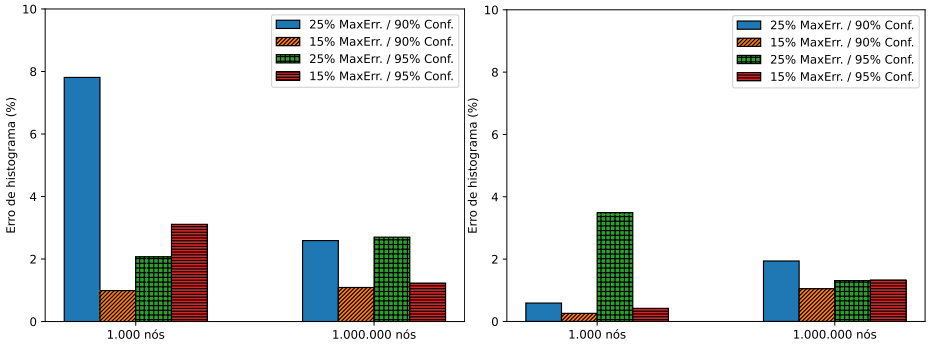
Os nossos resultados focam-se no número de saltos de necessários para obter uma estimativa e no erro do histograma calculado em cada experiencia. Na Figura 2 apresentamos o número de saltos de comunicação médio (arredondado às unidades), para as diversas combinações de parâmetros possíveis. Estes gráficos comparam o número de saltos entre experiências com tamanhos de redes diferentes. Do que é possível verificar, o número de saltos que o algoritmo necessita para obter um histograma não depende do tamanho do sistema. Note-se até que, em certos casos, redes de maior dimensão precisam de menos saltos, no entanto, isto deve-se à natureza probabilística do algoritmo. Outro aspeto relevante é a observação de que quando a distribuição de capacidades dos nós na rede não é uniforme, o algoritmo precisa de muitos mais saltos para obter uma estimativa, como visível na Figura 2b.

Na Figura 3 apresentamos o erro obtido no histograma estimado (no eixo do y) para todas as experiências (no eixo do x). Note-se que nestes gráficos



(a) Distribuição uniforme. (b) Distribuição log-normal.

Figura 2: Número de saltos de comunicação para a distribuição uniforme e log-normal.



(a) Distribuição uniforme. (b) Distribuição log-normal.

Figura 3: Erros de histograma para as distribuições uniforme e log-normal.

temos por objetivo comparar o erro do histograma dado os parâmetros utilizados pelo ResEst (margem de erro máxima e intervalo de confiança). Os histogramas calculados pelo algoritmo estão configurados para terem 5 classes e o seu erro é calculado pela soma das diferenças de cada uma das classes face ao histograma real dada a distribuição utilizada. A primeira coisa a notar é que o erro do histograma é superior nas experiências em que a margem de erro máxima é 25% quando comparado com experiências com margens de erro máximas de 15%. Isto acontece porque o passeio aleatório coleciona menos informação, o que pode ser verificado na Figura 2, onde as experiências com margens de erro máximas de 25% têm menos saltos de comunicação. Por outro lado, o ResEst consegue obter estimativas com erros similares independentemente do número de nós do sistema, sendo que na distribuição log-normal o erro sobe ligeiramente para as experiências com 1.000.000 nós, ao contrário de na distribuição uniforme, onde o número de nós tem menos impacto na taxa de erro resultante.

Variáveis			Resultados Médios de 10 Exp.			
Margem de Erro Max.	Intervalo de Conf.	Nº de Nós	Nº Hops	Média Obtida	Média Real	Erro do Histograma
25%	90%	1.000	15	53	50	7, 81%
25%	90%	1.000.000	13	52	50	2, 59%
15%	90%	1.000	39	51	50	0, 99%
15%	90%	1.000.000	38	50	50	1, 09%
25%	95%	1.000	23	49	50	2, 07%
25%	95%	1.000.000	20	52	50	2, 7%
15%	95%	1.000	53	52	50	3, 11%
15%	95%	1.000.000	54	50	50	1, 23%

Tabela 1: Resultados para Distribuição Uniforme.

Variáveis			Resultados Médios de 10 Exp.			
Margem de Erro Max.	Intervalo de Conf.	Nº de Nós	Nº Hops	Média Obtida	Média Real	Erro do Histograma
25%	90%	1.000	49	25	26	0, 59%
25%	90%	1.000.000	52	25	26	1, 94%
15%	90%	1.000	171	27	26	0, 26%
15%	90%	1.000.000	155	26	26	1, 05%
25%	95%	1.000	68	26	25	3, 49%
25%	95%	1.000.000	74	26	26	1, 31%
15%	95%	1.000	252	27	26	0, 42%
15%	95%	1.000.000	282	26	26	1, 33%

Tabela 2: Resultados para Distribuição Log-normal.

As Tabelas 1 e 2 sumarizam os nossos resultados experimentais para as duas distribuições de capacidade de nós utilizadas. Em ambas as tabelas, as primeiras três colunas referem o parâmetro e o respetivo valor variado na experiência. As quatro ultimas colunas referem os resultados obtidos para cada experiência. Estes incluem o número de saltos médio que um passeio aleatório fez, a média obtida pelo histograma calculado nos passeios aleatórios, a média real dada a distribuição dos nós, e por fim o erro do histograma calculado.

4.3 Análise de Resultados

É de notar que, independentemente da a distribuição das capacidades relativas dos nós, o nosso algoritmo é capaz de calcular histogramas estimados cujo erro é muito menor que a margem de erro máxima utilizada. Isto valida a nossa solução pelo facto de que, em intervalos de confiança de 90% e de 95%, o erro do histograma admitido poderia passar de 10% e 5% da margem máxima de erro, respetivamente. Ainda assim, os nossos resultados mostram que em nenhuma instância os erros dos histogramas ultrapassaram a margem de erro parameterizada. Estes resultados tão positivos devem-se ao facto de os erros dos histogramas serem muito menores do que os erros das médias estimadas e o critério de paragem estar dependente apenas do erro da média estimada.

É interessante verificar que o número de saltos necessários para se obter a estimativa não depende do número total de nós da rede. Ou seja, numa rede com 1.000 nós ou numa rede com 1.000.000 de nós, em que ambas as redes seguem a mesma distribuição de recursos, o número de saltos necessários para obter a estimativa é sensivelmente o mesmo (para os mesmos parâmetros de margem de erro e intervalo de confiança).

Assim, como pode ser observado pelos nossos resultados, o número de saltos necessário para se obter uma estimativa depende de três parâmetros: i) da margem de erro máxima utilizada pelo algoritmo — o que reparámos é que

quanto menor a margem de erro, mais saltos serão necessários para se obter a estimativa; *ii*) do intervalo de confiança desejado — quanto maior, mais saltos de comunicação serão necessários; e *iii*) da distribuição de recursos dos nós na rede — para distribuições uniformes menos saltos são necessários, para distribuições log-normal mais saltos serão necessários pois existe menos uniformidade na distribuição das capacidades.

Uma característica interessante deste algoritmo é que o processamento gasto por parte de cada nó para obter uma estimativa bastante fidedigna da rede é muito baixo. Por exemplo, para uma margem de erro máxima de 15% com um intervalo de confiança de 90% e uma distribuição de recursos uniforme, se o algoritmo for utilizado por cada nó a cada 5 minutos, cada nó vai enviar e receber entre 38 a 39 mensagens, em média, a cada 5 minutos. Como discutido anteriormente, este número não depende do número de nós na rede. Ou seja, neste caso, um determinado nó enviaria e receberia 38 a 39 mensagens quer a rede tenha 1.000 nós ou 1.000.000, fazendo deste algoritmo uma solução barata e escalável.

5 Conclusão

Neste artigo apresentámos um algoritmo descentralizado para estimar o quão poderoso um nó é relativamente ao resto da rede, através da obtenção de um histograma que codifica a distribuição da capacidade de todos os nós existentes. A nossa solução — ResEst — é um novo algoritmo descentralizado cujo custo de operação médio por nó não depende do tamanho do sistema, mas sim dos parametros de qualidade dessa estimativa e da distribuição global de resursos na rede. A nossa validação experimental, realizada por simulação, mostra a validade da solução, e ilustra o seu baixo custo e fiabilidade.

Dada a potencialidade da solução, os próximos passos serão aplicar este algoritmo distribuído em casos de uso práticos — soluções onde os nós tenham de saber quão poderosos são em relação ao resto da rede, e.g., soluções de autobalanceamento de carga pelas diversas máquinas de uma determinada rede. A aplicação do ResEst nestas soluções vai permitir a validação deste algoritmo em emulação, ou seja, tendo em consideração todas as propriedades e restrições das redes físicas.

Referências

1. Web 3.0 technology stack. https://web3.foundation/about/, accessed July 2021
2. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed computing **18**(4) (2006)
3. Baquero, C., Almeida, P.S., Menezes, R., Jesus, P.: Extrema propagation: Fast distributed estimation of sums and network sizes. IEEE Transactions on Parallel and Distributed Systems **23**(4) (2012)

4. Bawa, M., Garcia-Molina, H., Gionis, A., Motwani, R.: Estimating aggregates on a peer-to-peer network. Technical Report 2003-24, Stanford InfoLab (April 2003), http://ilpubs.stanford.edu:8090/586/
5. Benet, J.: IPFS - Content Addressed, Versioned, P2P File System. Tech. Rep. Draft 3 (2014), http://arxiv.org/abs/1407.3561
6. Costa, P.A., Fouto, P., Leitao, J.: Overlay networks for edge management. In: 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA) (2020)
7. Eugster, P.T., Guerraoui, R., Kermarrec, A.., Massoulie, L.: Epidemic information dissemination in distributed systems. Computer **37**(5) (2004)
8. Fernandez, A., Gramoli, V., Jimenez, E., Kermarrec, A.M., Raynal, M.: Distributed slicing in dynamic systems. In: 27th International Conference on Distributed Computing Systems (ICDCS '07) (2007)
9. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.: Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In: Crowcroft, J., Hofmann, M. (eds.) Networked Group Communication (2001)
10. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: Wait-free coordination for internet-scale systems. In: USENIX ATC. vol. 8 (2010)
11. Jelasity, M., Kermarrec, A.M.: Ordered slicing of very large-scale overlay networks. In: Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06). IEEE (2006)
12. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., Van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems (TOCS) **25**(3) (2007)
13. Kermarrec, A.., Massoulie, L., Ganesh, A.J.: Probabilistic reliable dissemination in large-scale systems. IEEE Transactions on Parallel and Distributed Systems **14**(3) (2003)
14. Leitão, J.: Gossip-Based Broadcast Protocols. Master's thesis, Faculdade de Ciências da Universidade de Lisboa (2007)
15. Leitao, J., Pereira, J., Rodrigues, L.: Hyparview: A membership protocol for reliable gossip-based broadcast. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07) (2007)
16. Manikandan, S.G., Ravi, S.: Big data analysis using apache hadoop. In: 2014 International Conference on IT Convergence and Security (ICITCS) (2014)
17. Massoulié, L., Le Merrer, E., Kermarrec, A.M., Ganesh, A.: Peer counting and sampling in overlay networks: Random walk methods. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing (2006)
18. Rodrigues, R., Druschel, P.: Peer-to-peer systems. Communications of the ACM **53**(10) (2010)
19. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: New aggregation techniques for sensor networks. In: Proc. of SenSys '04 (2004)
20. Sudo, Y., Shibata, M., Nakamura, J., Kim, Y., Masuzawa, T.: Self-stabilizing population protocols with global knowledge. IEEE Transactions on Parallel and Distributed Systems **32**(12) (2021)
21. Valerie J. Easton, J.H.M.: Statistics glossary v1.1. http://www.stats.gla.ac.uk/steps/glossary/ (September 1997)
22. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Tech. rep. (2014)
23. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I., et al.: Spark: Cluster computing with working sets. HotCloud **10**(10-10) (2010)