Cálculo de Sub-Redes e Pontos de Articulação

Relatório do 1º Projecto - Análise e Síntese de Algoritmos

Baltasar Dinis, 89416 e Afonso Ribeiro, 86752

***Resumo*— Uma rede deve ser desenhada de forma a que seja resiliente a falhas de componentes individuais, permitindo ao seu utilizador uma utilização contínua do sistema. A existência de pontos únicos de falha[1] compromete esta resiliência, impedindo que a rede consiga escalar face ao tráfego. Neste relatório, expomos uma metodologia para automaticamente verificar se uma rede é resiliente e quantas sub-redes estão presentes na mesma. Este trabalho foi realizado no contexto da Unidade Curricular de Análise e Síntese de Algoritmos, no ano lectivo de 2018-2019.**

I. INTRODUÇÃO

Redes de computadores são hoje em dia ubíquas e essenciais para o funcionamento normal de empresas, do Estado e da sociedade em geral. Devido à sua importância, uma rede deve ser capaz de tolerar falhas individuais e independentes dos seus componentes, de forma a que um serviço que a utilize consiga escalar com o tráfego. Se a rede tiver um ponto que, ao falhar, desconecte a rede, não se pode considerar que a rede tem uma topologia resiliente. É por isso importante conseguir, de forma expedita, identificar estes pontos.

É também útil conseguir saber quais são as sub-redes existentes numa dada topologia de roteadores, e o que aconteceria caso esses pontos falhassem simultaneamente.

Neste relatório, apresentaremos uma solução para este problema. A estrutura do relatório é a seguinte: em II, fazemos um mapeamento do problema para um problema de grafos e expomos a nossa solução; em III, fazemos uma análise teórica do algoritmo empregue, nomeadamente em termos de complexidade; em IV apresentamos os resultados da nossa avaliação experimental e em V comentamos a correspondência entre os resultados experimentais e os valores teóricos.

[1]*Single Points of Failure* em inglês

II. DESCRIÇÃO DA SOLUÇÃO

Representamos uma rede de roteadores como um grafo não dirigido. Notemos que uma sub-rede é uma componente do grafo, pois existem caminhos entre dois quaisquer roteadores de uma sub-rede, e não existe nenhum roteador atingível a partir de um outro na sub-rede que não pertença ele mesmo à sub-rede.

Mapeamos conceptualmente os pontos únicos de falha na noção de ponto de articulação. Na literatura, [1], define-se um ponto de articulação como aquele cuja remoção causaria uma desconexão de um grafo conexo (consideramos as componentes como sub-grafos conexos). Note-se a seguinte, e interessante, propriedade dos pontos de articulação:

Lema. Seja $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ um grafo e $a \in \mathscr{V}$. Se existir um par de vértices $u, v \in \mathscr{V}$ tal que existe um caminho de u para v que passa por a. Então a é um ponto de articulação se e só se todos os caminhos de u para v passarem por a.

Dem. Se existir um caminho de u para v que não passe por a então a remoção de a não faz com que a componente se desconecte. Se a é um ponto de articulação então a sua remoção terá de desconectar a componente. Isto só acontece se todos os caminhos que vão de u para v passarem por a (caso contrário a componente manter-se-ia conexa).

Note-se que esta propriedade é importante (em [2] é dada como a definição de ponto de articulação) definindo a essência do ponto de articulação como ponto único de falha.

A nossa abordagem baseia-se na procura em profundidade[2], empregando algoritmos definidos por Tarjan [2], [3]. Em cada uma das subsecções seguintes apresentamos como a DFS é aplicada para produzir cada um dos

A. Cálculo de Sub-Redes

Uma sub-rede é uma componente do grafo. Como descrito em [3], podemos obter as componentes aplicando a DFS. A partir de um vértice qualquer, começamos a fazer a procura. Quando não existirem mais vértices a explorar encontramos todos os vértices da componente.

No contexto do problema, não temos que identificar a componente inteira, mas sim o identificador do roteador com maior índice. Fazemos isso de forma eficiente impondo uma ordenação na extracção. Extraímos o roteador não visitado de maior índice, adicionando-o ao fim de uma lista que guarda os identificadores das sub-redes, pois é garantido que todos os roteadores de índice maior que o dele já tenham sido explorados - não sendo possível que façam parte desta componente.

Como temos de imprimir a lista de identificadores de sub-rede por ordem crescente, percorremos a lista em ordem inversa.

B. Cálculo de Pontos de Articulação

Introduzimos os seguintes lemas .

Lema. Se $v \in \mathscr{V}$ tiver menos de duas adjacências não é um ponto de articulação. A demonstração é trivial.

Lema. Seja $root \in \mathscr{V}$ a raiz de uma árvore gerada pela DFS. $root$ é um ponto de articulação se e só se tem pelo menos duas crianças na árvore.

[2] *Depth-First Search*, ou DFS, em inglês

Dem. Se não tiver pelo menos duas adjacências (pelo lema anterior) não pode ser ponto de articulação. Se tiver n crianças é garantido que não hajam ligações entre as sub-árvores (se existissem, não seriam sub-árvores distintas, pois só se escolhe o filho da raiz após a árvore estar completamente explorada). Logo a raiz tem de ser o único nó a ligar as sub-árvores, sendo um ponto de articulação.

Def. (Aresta para Trás) Dizemos que (u,v) é uma aresta para trás numa DFS quando tanto u como v já foram descobertos, u é descendente de v e está a ser tratada a aresta.

Para calcular as arestas para trás, mantemos uma lista dos valores de $low : \mathscr{V} \to \mathbb{N}$, para além da usual lista de valores de descoberta (d) e de pais (π). $low(v)$ é o mínimo entre o tempo da sua descoberta e o low dos seus descendentes directos. Isto significa que, se existir uma aresta para trás (u,w), onde u é descendente de v e w antecessor, então $low(u) = low(w)$. Assim, conseguimos descobrir se um dos filhos de v (ou algum dos seus descendentes) tem arestas para um dos antecessores de v.

Podemos sintetizar dois predicados, um para dados dois vértices descobrir se existe uma aresta para trás (e subsequentemente actualizar o low) e outro para, dada uma sub-árvore DFS (completamente explorada), descobrir se há uma aresta para trás da raiz da sub-árvore.

$$\begin{aligned} is\_back\_edge(u,w) &\leftarrow \\ d(w) < d(u) \wedge \pi(u) \neq w & \\ subtree\_has\_backedge(root) &\leftarrow \\ \exists c \in Children(root) : & \\ low(c) < low(root) & \end{aligned}$$

Com base nisto, podemos verificar se um ponto é de articulação ou não utilizando o seguinte predicado.

$$\begin{aligned} &is\_articulation\_point(v) \leftarrow \\ &(is\_dfs\_root(v) \land n\_children(v) \geq 2) \lor \\ &(\neg is\_dfs\_root(v) \land \exists u \in Children(v) : \\ &\neg subtree\_has\_backedge(u)) \end{aligned}$$

Fazemos este cálculo para todos os nós à medida que se faz a pesquisa (mantendo uma estrutura de booleanos que indica se eles são pontos de articulação), calculando o predicado se o nó ainda não for ponto de articulação.

C. *Componente Máxima*

Depois de remover os nós (marcando-os com uma cor especial), calculamos uma versão simplificada da DFS, onde mantemos apenas o tamanho da maior componente.

III. ANÁLISE TEÓRICA

A nossa implementação tem 4 passos relevantes:

1) Construir o grafo;
2) Aplicar uma DFS, como descrito em II-B para calcular os pontos de articulação;
3) Remover os pontos de articulação;
4) Tornar a aplicar uma DFS, como descrito em II-C, para calcular o tamanho da maior sub-rede.

Notemos que, uma vez calculados os pontos de articulação, removê-los é uma operação trivial, completada em $\Theta(V)$, no pior caso (basta marcá-los com uma cor especial).

A. *Representação do Grafo*

Representamos o grafo por uma lista de adjacências. Notemos que esta representação é $O(V+E)$ [1] em uso de memória e que a operação de obter a lista e adjacências de um nó é $O(1)$.

B. *Construção do Grafo*

O input é dado em 3 secções: 1) Uma linha com o número de roteadores; 2) Uma linha com o número de ligações; 3) lista de ligações. Logo a leitura é feita em $\Theta(E)$.

Como a adição de um nó no grafo é executada em tempo constante e a adição de uma aresta também, concluímos que a construção do grafo é $\Theta(V+E)$.

C. *Pesquisa em Profundidade*

Aplicar uma DFS é tem complexidade temporal $\Theta(V+E)$ [1]. Mostramos agora que a nossas adaptações da pesquisa limitam-se a fazer computações em tempo constante, e que por isso o nosso algoritmo adaptado é também um $\Theta(V+E)$.

Na primeira aplicação, que calcula os pontos de articulação e os identificadores das sub-redes, são feitas as seguintes operações:

- Inicializações: $\Theta(1)$
- Obtenção da lista nós adjacentes: $\Theta(1)$

  Para cada adjacente não descoberto
- Incrementação do número de filhos: $\Theta(1)$
- Inicialização do valor do pai: $\Theta(1)$
- Actualização do *low* (comparação e eventual atribuição) : $\Theta(1)$
- Verificação se é ponto de articulação : $\Theta(1)$, pois as dependências do predicado apresentado em II-B já se encontram calculadas, ou são calculáveis em $\Theta(1)$

  Para cada adjacente (fora o pai) não descoberto:
- Actualização do *low*: $\Theta(1)$

Na segunda aplicação a visita é mais simples:

- Inicialização da cor: $\Theta(1)$
- Obtenção da lista nós adjacentes: $\Theta(1)$

  Para cada adjacente não descoberto
- Incrementação do número de nós da sub-rede: $\Theta(1)$

Como todas as operações da visita não estruturais à pesquisa em profundidade são $\Theta(1)$ e a própria pesquisa é $\Theta(V+E)$, concluímos que ambas as aplicações têm uma complexidade $\Theta(V+E)$.

D. Avaliação do Algoritmo Proposto

Concluímos que o algoritmo tem complexidade:

$$\begin{aligned}
&\Theta(V+E) && \text{(Construção do Grafo)}\\
&+\Theta(V+E) && \text{(Aplicação 1ª DFS)}\\
&+\Theta(V) && \text{(Rem. P. de Articulação)}\\
&+\Theta(V+E) && \text{(Aplicação 2ª DFS)}\\
&= 3\Theta(V+E)+\Theta(V) = && \Theta(V+E)
\end{aligned}$$

IV. AVALIAÇÃO EXPERIMENTAL

Para a avaliação experimental, geramos testes com as seguintes variantes.

- V: número de roteadores na rede
- E: número de ligações na rede
- S: número de sub-redes

Consideramos mais interessante o caso onde $E > V$, pois no caso contrário há pelo menos $V-(2*E)$ vértices sozinhos. Desta forma, conseguimos redes mais complexas e, por conseguinte, aplicações não triviais do algoritmo.

Na Figura 1 está exposta a relação entre o número de arestas e a complexidade temporal (medida em número de ciclos de relógio). A recta de regressão linear encontrada tem um coeficiente de determinação $R^2 = 0.897$, pelo que concluímos que o desempenho da nossa solução varia linearmente com o número de arestas do grafo, estando por isso em concordância com a análise teórica.

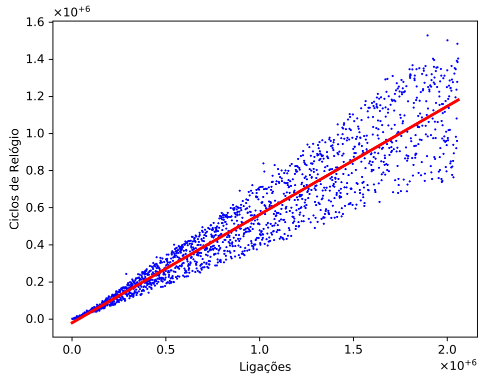

Fig. 1. Resultados experimentais: Latência em função do número de arestas

Os testes foram efectuados numa arquitectura Intel® Core™i5-6200U. O código foi instrumentado directamente, através da inclusão de duas instruções, uma para obter o valor do relógio no momento inicial e outra no momento final.

V. CONCLUSÃO

Apresentámos uma solução para o problema de descoberta de pontos de falha única numa rede, bem como uma simulação do pior caso para a rede - a falha simultânea de todos os pontos de falha única. Para tal, mapeamos o problema para um problema de grafos, onde procuramos os pontos de articulação. Para tal, recorremos a uma variante da pesquisa em profundidade. Para calcular o tamanho da maior componente, limitamo-nos a aplicar de novo a pesquisa em profundidade. Mostramos teoricamente e confirmamos experimentalmente esses resultados, onde conseguimos que a solução tivesse complexidade $\Theta(V+E)$.

REFERÊNCIAS

[1] T. Cormen, C. Leiserson e L. R. Rivest, *Introduction to Algorithms* 1ª edição. Cambridge, Massachussets: The MIT Press, 1990.

[2] R. E. Tarjan. "Depth-First Search and Linear Graph Algorithms" in SIAM Journal on Computing, Vol. 1, No. 2, June 1972.

[3] J. Hopcroft and R. E. Tarjan. "Efficient Algorithms for Graph Manipulation" in Communications of the ACM, Vol. 16, No. 6, 1973.