Sistemas Distribuídos em Larga Escala
Trabalho Prático

-

Mestrado em Engenharia Informática
Universidade do Minho

Grupo nº 1

PG41080	João Ribeiro Imperadeiro
PG41081	José Alberto Martins Boticas
PG41091	Nelson José Dias Teixeira

26 de maio de 2020

1 Introdução

O presente relatório descreve o desenvolvimento do projeto de cariz prático da unidade curricular de Sistemas Distribuídos em Larga Escala. Neste trabalho é requerida a implementação de um dos algoritmos de agregação distribuída disponíveis no documento de suporte ao enunciado do mesmo [2]. Após a implementação do algoritmo escolhido, é posteriormente solicitado o teste do mesmo no simulador desenvolvido ao longo do semestre do presente ano letivo.

Tal como é mencionado nas referências bibliográficas [1] e [2], a agregação de dados distribuídos desempenha um papel bastante importante na concepção de diversos sistemas escaláveis uma vez que possibilita a determinação descentralizada de propriedades globais significativas, que posteriormente podem ser utilizadas para direcionar a execução de outras aplicações distribuídas. Vários algoritmos de agregação distribuída foram propostos ao longo dos últimos anos, exibindo propriedades diferentes em termos de precisão, desempenho e de métricas de comunicação. No entanto, muitas dessas abordagens não possuem caraterísticas relacionadas com a tolerância de faltas. Desta forma, no âmbito de sistemas distribuídos, este grupo de trabalho viu-se interessado em implementar um dos algoritmos com esta propriedade.

Relativamente à estrutura deste relatório, é exibido o algoritmo escolhido, apresentando o conceito e a implementação intrínsecos ao mesmo. Para além disso, são evidenciados alguns aspetos relativos

ao simulador utilizado para proceder à realização de testes do algoritmo em causa, efetuando finalmente uma análise dos resultados obtidos.

2 Algoritmo

Dos algoritmos de agregação distribuída presentes no documento referenciado no enunciado deste trabalho prático [2], optou-se pela escolha do algoritmo ***flow updating***. Este, ao nível de comunicação, é classificado como não estruturado, inserindo-se na categoria *gossip* que, por sua vez, diz respeito à forma como as mensagens são disseminadas pela rede de comunicação. Quanto à perspetiva computacional, este algoritmo tem como objetivo a computação iterativa de médias parciais que, ao longo do tempo, convergem para um determinado valor. Esta última técnica permite também a derivação de outras funções de agregação (como por exemplo, *count* ou *sum*), de acordo com as combinações dos valores inicialmente instanciados.

Uma das razões que levou este grupo a escolher este algoritmo foi a capacidade do mesmo em tolerar a injeção de faltas transientes. Esta última caraterística é bastante importante sobretudo no que diz respeito à perda de mensagens trocadas na rede de comunicação. Com pequenas alterações, é ainda possível que o algoritmo suporte falhas de membros, falhas permanentes de ligações e mudanças de vizinhança. Para além destas vantagens associadas ao contexto de sistemas distribuídos, este algoritmo possui não só um melhor desempenho quando comparado com os outros da mesma classe, como também possibilita uma computação mais precisa de valores. Por fim, a execução do algoritmo em causa é independente da topologia do roteamento de rede.

No 4º capítulo deste documento são devidamente clarificados todos os resultados obtidos após a implementação do algoritmo escolhido, discutindo-se a veracidade das propriedades mencionadas acima.

2.1 Conceito

O algoritmo *flow updating* difere de outros propostos para o mesmo problema no sentido em que não se procede ao envio de mensagens complexas que, posteriormente, são armazenadas e disseminadas. Em contraste, faz uso do conceito de fluxo (no sentido da teoria de grafos), atualizando-o com recurso ao valor inicial e à

contribuição de fluxos de outros nós vizinhos, sendo que o fluxo é simétrico nos extremos de cada ligação. Por isto, a soma das médias de cada um dos nós deve manter-se constante. Assim, partilha-se apenas o fluxo com os vizinhos, sendo que a perda de uma destas mensagens não implica uma falha grave. A intuição é que a simetria pode ser quebrada, se for possível garantir que uma mensagem chegará no futuro e a restabeleça.

Importa realçar que existem duas versões do algoritmo. A primeira assenta no envio de mensagens com recurso a *broadcast* (N para N) enquanto que a segunda enquadra-se no envio de mensagens em *unicast* (1 para 1).

2.2 Implementação

Tal como foi referido no fim do subcapítulo anterior, foram implementadas duas versões distintas do algoritmo em questão. À semelhença do que foi elaborado durantes as aulas práticas desta unidade curricular, estas duas versões foram implementadas na linguagem `Python`. Para cada uma das mesmas são exibidos os respetivos detalhes computacionais e, ainda, um pequeno esquema relativo ao seu pseudocódigo.

2.2.1 *Broadcast*

Relativamente a esta abordagem, criou-se uma classe para representar um nó do grafo, com o nome `flowUpdatingBroad`, que contém as seguintes informações no seu estado:

- um dicionário f que contém os fluxos desses nós para os seus vizinhos;
- um dicionário e que contém as estimativas que recebeu dos seus vizinhos;
- o seu valor inicial v;
- a lista dos seus vizinhos;
- as mensagens acumuladas de uma ronda;
- um inteiro *timeout* que representa a duração pré-estabelecida de cada ronda.

Esta classe contém ainda as seguintes funções:

- `gen_message` - devolve a lista de mensagens a serem enviadas nessa ronda, incluindo uma mensagem especial de *timeout*, dirigida ao próprio nó e que representa o fim de uma ronda;

- `state_transition` - executada em reação à mensagem de *timeout*, ou seja, no fim da ronda, por forma a calcular as novas estimativas e fluxos dos vizinhos, bem como os próprios;
- `calculate_estimate` - executada pelo método anterior e que corresponde ao valor calculado durante a transição de estado, ou seja, devolve a nova estimativa para o valor objetivo.

Dito isto, divulga-se de seguida uma figura representativa do pseudocódigo associado a esta versão:

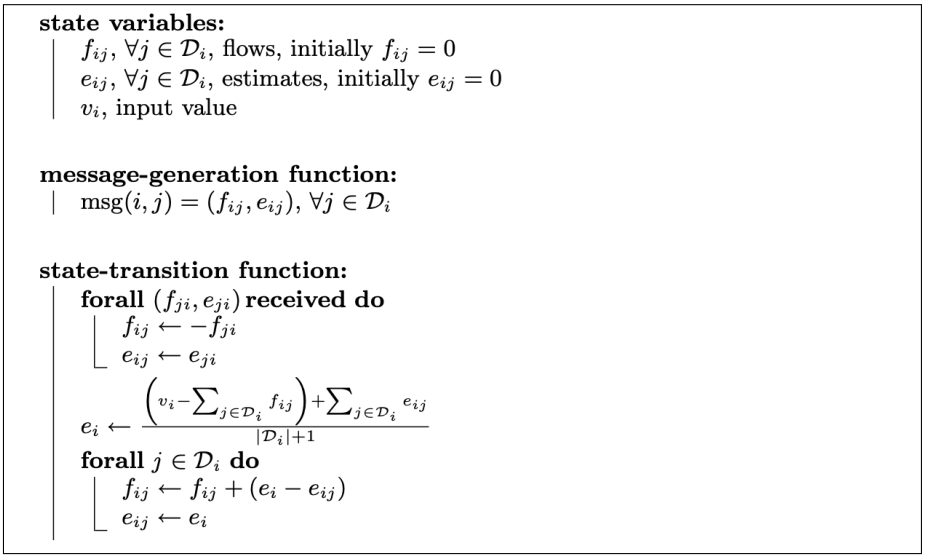

Figura 1: Pseudocódigo do algoritmo *Flow Updating* (*broadcast*)

2.2.2 *Unicast*

Quanto a esta versão, concebeu-se, da mesma forma, uma classe para representar um nó do grafo, com o nome `flowUpdatingUni`, que contém as mesmas informações, no seu estado, que as indicadas acima, para a classe de *Broadcast*. Para além destas, têm ainda um inteiro representativo do identificador do vizinho escolhido, sendo que o mesmo é selecionado pela função `chooseNeighbor`, que, tal como o nome transparece, devolve aleatoriamente o identificador de um dos vizinhos.

As funções desta classe são as mesmas que as expostas anteriormente na classe *broadcast*, com as seguintes alterações:

- `gen_message` - calcula apenas uma mensagem para o vizinho k, em vez de todos os vizinhos, e a mensagem especial de fim de ronda;

- `state_transition` - calcula apenas a estimativa e o fluxo do vizinho k;

Expõe-se agora um diagrama alusivo ao pseudocódigo desta versão:

state variables:
- $f_{ij}, \forall j \in \mathcal{D}_i$, flows, initially $f_{ij} = 0$
- $e_{ij}, \forall j \in \mathcal{D}_i$, estimates, initially $e_{ij} = 0$
- v_i, input value
- k, chosen neighbor

message-generation function:
- $\text{msg}(i, k) = (f_{ik}, e_{ik})$

state-transition function:
- **forall** (f_{ji}, e_{ji}) **received do**
  - $f_{ij} \leftarrow -f_{ji}$
  - $e_{ij} \leftarrow e_{ji}$
- $e_i \leftarrow \frac{\left(v_i - \sum_{j \in \mathcal{D}_i} f_{ij}\right) + \sum_{j \in \mathcal{D}_i} e_{ij}}{|\mathcal{D}_i| + 1}$
- $k \leftarrow chooseNeighbor(\mathcal{D}_i)$;
- $f_{ik} \leftarrow f_{ik} + (e_i - e_{ik})$
- $e_{ik} \leftarrow e_i$

Figura 2: Pseudocódigo do algoritmo *Flow Updating* (*unicast*)

3 Simulador

No que toca ao simulador utilizado, para proceder à realização de testes sobre o algoritmo escolhido, foram adicionados algumas funcionalidades face ao que foi desenvolvido durante as aulas da componenete prática desta unidade curricular. Dado que o algoritmo selecionado suporta a injeção de faltas transientes, foi incorporado no simulador deste trabalho um mecanismo associado a esta última propriedade. Como tal, foi declarada uma variável *fp* relativa à probabilidade de ocorrência de faltas, isto é, à probabilidade de uma determinada mensagem se perder na rede de comunicação. Para além desta caraterística, foi também introduzida uma técnica de mudança de vizinhança. Tal como a última propriedade, foi instanciada uma incógnita *vcp* relativa ao número de segundos até que surja uma nova mudança de vizinhança no grafo em causa.

Com estas modificações, foi possível averiguar a natureza inerente ao algoritmo *flow updating*, observando-se o seu comportamento no contexto de sistemas distribuídos.

4 Análise de resultados obtidos

Os resultados da avaliação obtidos, quando comparados com outras abordagens de *averaging*, revelaram que os superam em termos de complexidade temporal (maior desempenho) e ao nível do número de mensagens trocadas (*overhead*).

5 Conclusão

Este algoritmo tolera a perda substancial de mensagens (*link failures*), enquanto outros algoritmos concorrentes da mesma categoria podem ser afetados por uma única mensagem perdida.

Referências

[1] Paulo Jesus, Carlos Baquero e Paulo Almeida. «Fault-Tolerant Aggregation by Flow Updating». Em: jun. de 2009, pp. 73–86. DOI: 10.1007/978-3-642-02164-0_6.

[2] Paulo Jesus, Carlos Baquero e Paulo Almeida. «A Survey of Distributed Data Aggregation Algorithms». Em: *Communications Surveys & Tutorials, IEEE* 17 (out. de 2011). DOI: 10.1109/COMST.2014.2354398. URL: https://arxiv.org/pdf/1110.0725.pdf.