Sistemas Distribuídos em Larga Escala

Trabalho Prático

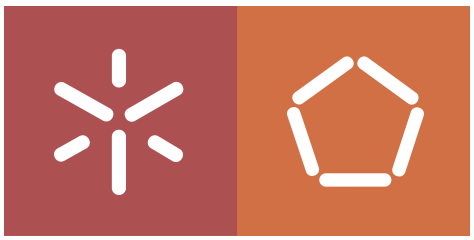
-

Mestrado em Engenharia Informática

Universidade do Minho

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

31 de maio de 2020



Conteúdo

Capítulo 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.

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, este grupo de trabalho viu-se interessado em implementar um dos algoritmos com esta propriedade, *flow updating* [1].

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-se posteriormente uma análise dos resultados obtidos. Por fim, são resumidos os tópicos referentes a este trabalho, concluindo, assim, a sua realização.

Capítulo 2

Algoritmo

O algoritmo ***flow updating***, 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. 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 e falhas permanentes de ligações. 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.

2.2.1 *Broadcast*

Relativamente a esta abordagem, criou-se uma classe para representar um nó do grafo, com o nome `flowUpdatingBroadcast`, 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.

2.2.2 *Unicast*

Quanto a esta versão, concebeu-se, da mesma forma, uma classe para representar um nó do grafo, com o nome `flowUpdatingUnicast`, 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;

Capítulo 3

Simulador

O simulador presente neste projeto consiste num programa escrito em `Python` que modela um sistema distribuído composto por um número arbitrário de nós e respetivas ligações entre eles. Para além disso, este simula a passagem de tempo, permitindo não só a injeção de faltas através da eliminação de mensagens como também possibilita a mudança de vizinhança.

Para poder simular um sistema composto por vários nós, este guarda no seu estado uma lista com os nós do sistema e as distâncias de todas as ligações que existem. Para além disto, tem uma lista onde guarda as mensagens pendentes do sistema e um inteiro que representa o tempo atual (numa unidade arbitrária).

De forma a ser possível costumizar o comportamento da simulação, podem ser configurados diversos parâmetros, entre os quais a probabilidade de falha no envio de uma mensagem, a periodicidade em que ocorrem mudanças de vista (isto é, a alteração das ligações do sistema) e, ainda, o *timeout* que é utilizado para o envio de

mensagens periódicas de um nó para ele mesmo (representa, por exemplo, o fim de uma ronda).

Para ajustar ao algoritmo selecionado, tivemos de implementar um mecanismo de deteção da precisão da estimativa atual dos nós do grafo gerado, por forma a garantir que o algoritmo converge corretamente. Para isso, foi desenvolvida uma forma de computar o valor alvo (*target*) para que, posteriormente, este seja comparado com a estimativa momentânea de todos os nós. No caso de haver uma aproximação com um erro inferior a 0.01, dá-se por concluída a execução do algoritmo.

Quanto aos métodos que o simulador possui, temos a destacar:

- `start` - gera as mensagens iniciais, calcula o valor objetivo e programa a primeira mudança de vista, no caso desta existir;
- `runLoop` - processa as mensagens do sistema, tratando não só de entregá-las ao seu destino como também de incorporar as novas. Para além disso, é responsável pelas mudanças de vista e injeção de faltas, bem como a terminação do algoritmo no caso do valor objetivo ser atingido (dentro da margem de erro estipulada);
- `closeToTarget` - verifica se a estimativa do sistema está suficientemente próxima do valor objetivo.

Assim, foi possível averiguar a natureza inerente ao algoritmo *flow updating*, observando-se o seu comportamento no âmbito de sistemas distribuídos.

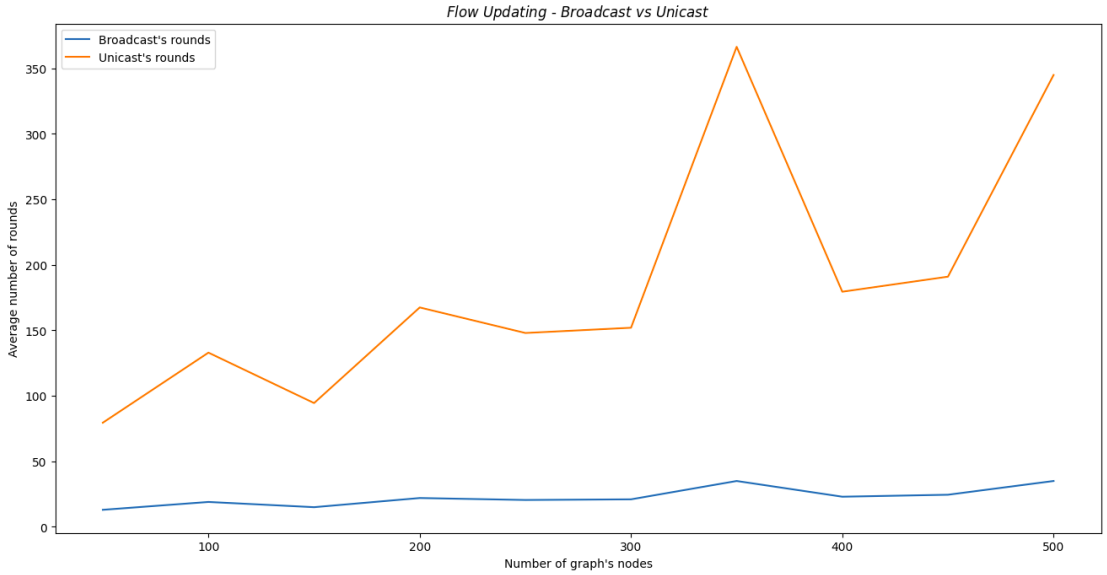
Capítulo 4

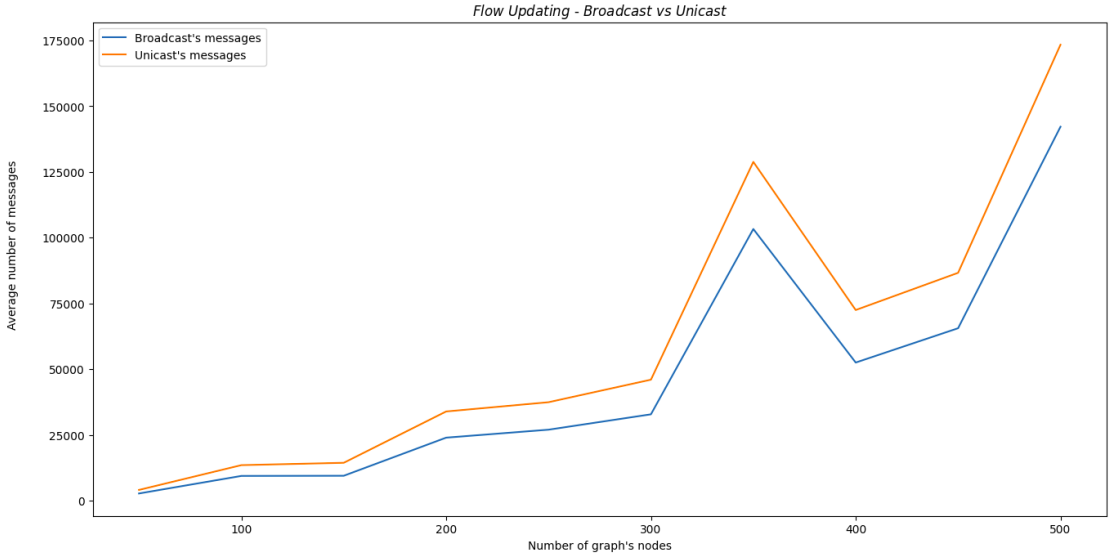
Análise de resultados obtidos

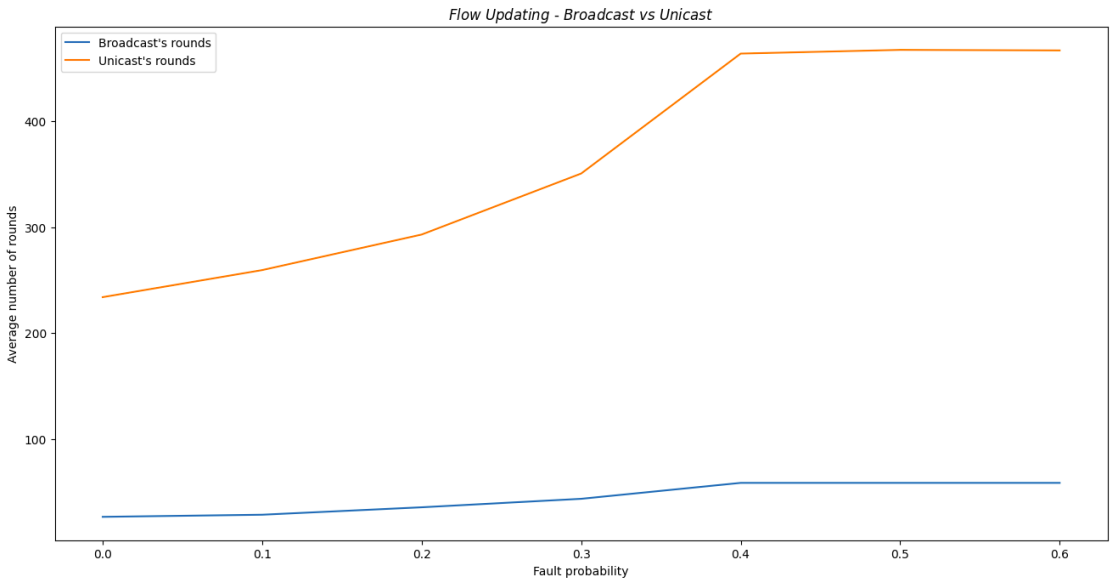
Em relação às duas variantes deste algoritmo, verificou-se que a versão *broadcast* é melhor do que a *unicast* em termos do número de rondas e de mensagens trocadas. O número de rondas apresenta uma maior descrepância entre as duas versões, o que é explicável pelo facto de na versão *broadcast* cada nó enviar mensagens para todos os vizinhos em cada ronda, enquanto na versão *unicast* cada nó envia apenas um mensagem para um vizinho a cada ronda. É também de salientar que mesmo em condições de perdas significativas de mensagens, o algoritmo elegido exibe um desempenho satisfatório, convergindo sempre para o valor pretendido.

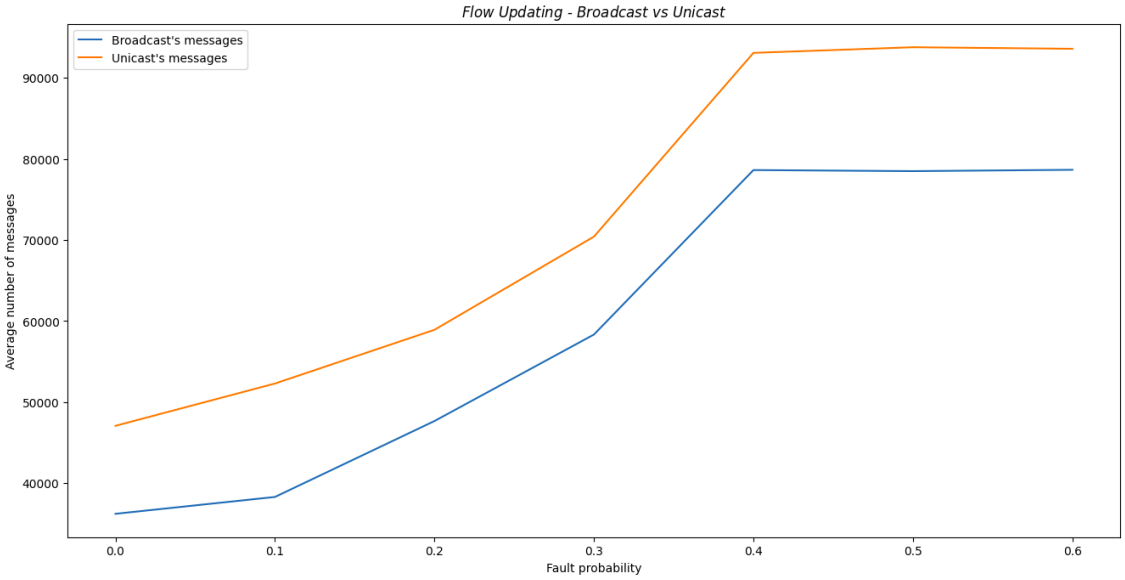
Nos gráficos em que podemos ver a relação entre o número de nós e o número de mensagens/rondas temos algumas flutuações em relação aos resultados que esperávamos. Tal pode ser explicado por serem criados grafos novos para cada número diferente de nós, sendo que, por ser seguido um método aleatório na geração das ligações do grafo, o número de ligações ou até mesmo o diâmetro do grafo, podem ter influência no tempo de execução do algoritmo. Nos gráficos que relacionam a percentagem de mensagens perdidas com o número de mensagens/rondas as mesmas flutuações já não existem, uma vez que o grafo utilizado é sempre o mesmo.

Exibem-se de seguida 4 gráficos que evidenciam a respetiva análise:


Flow Updating - Broadcast vs Unicast
Broadcast's rounds
Unicast's rounds
Average number of rounds
Number of graph's nodes


Flow Updating - Broadcast vs Unicast
Broadcast's messages
Unicast's messages
Average number of messages
Number of graph's nodes


Flow Updating - Broadcast vs Unicast
Broadcast's rounds
Unicast's rounds
Average number of rounds
Fault probability



Capítulo 5

Conclusão

Apesar do algoritmo selecionado por este grupo de trabalho trocar um maior número de mensagens na rede de comunicação (ainda que pouco significativo) do que outros de agregação distribuída, *flow updating* é a única abordagem robusta e resiliente de agregação tolerante a falhas conhecida. Esta é capaz de se adaptar continuamente às mudanças topológicas da rede, sem recorrer a nenhum tipo de mecanismo de reinicialização. Além disso, este algoritmo é adequado em cenários dinâmicos, onde são necessárias estimativas precisas sem haver a imposição muito restrita ao nível da quantidade de mensagens trocadas. Durante a implementação do algoritmo, o simulador ajudou na conceção do mesmo, uma vez que foi possível tirar ilações enquanto este era desenvolvido. Também permitiu encontrar erros, testar o algoritmo, comparar versões, testar otimizações e implementá-las. No geral, o desempenho do simulador e do algoritmo são satisfatórios, o que nos permite afirmar que atingimos os objetivos propostos.

Bibliografia

[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.