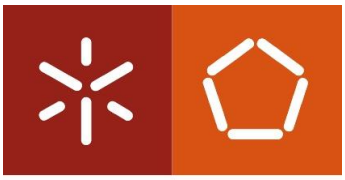

Universidade do Minho

Departamento de Informática

Sistemas Distribuídos em Larga Escala

João Vilaça (a82339), Eduardo Jorge (a83344)

5 de Março de 2021

1 Introdução

No âmbito dos sistemas distribuídos, é comum a necessidade de processamento de dados de forma paralela e simultânea nos vários processos que o compõe, desde tarefas simples, como a aplicação da função soma a um vector de números, até ao processamento de imagens para algoritmos de machine learning. Em todos estes casos, é fácil de perceber como decompor estas tarefas para que as mesmas possam ser resolvidas em pequenas partes e mais tarde agregadas num único resultado, mas não é assim tão directo escolher, implementar e analisar uma arquitectura de comunicação entre estes processos distribuídos que tenha bons resultados. Neste trabalho, desenvolve-se um simulador, capaz de avaliar algoritmos das várias classes de agregação, com facilidade de replicar as várias condições de um sistema real, entre as quais, o número de nodos, condições de rede no que diz respeito a perde de mensagens e mudanças de views.

2 Breve descrição do algoritmo

O algoritmo escolhido tira partido da redundância de caminho introduzida pelo uso de vários pais para melhorar a robustez do esquema de agregação (tolerância à perda de mensagens). Este algoritmo tem melhor grau de acerto que outros algoritmos "tree-based"mas também sobrecarregam muito mais a rede.

O DAG permite que os nós possuam pais alternativos, as mensagens "Request"contém uma lista de nós pais, permitindo que os filhos saibam os pais dos seus pais (avós). As mensagens de resposta são enviadas para cima pelos vários caminhos alternativos e podem ser agregadas, encaminhadas ou descartadas.

Apesar de tudo, esta abordagem não supera completamente o problema de perda de mensagens das topologias em árvore, tudo depende da maneira como o DAG está construído, pois os nodos podem só ter apenas um pai, e se a mensagem falhar nessa ligação, nunca chegará ao destino.

As mensagens são agregadas se o nó receptor corresponder ao destino, encaminhadas se o destino for o pai de um nó e descartadas de outra forma.

3 Motivação para a escolha do algoritmo

Uma das principais classes de algoritmos de agregação é baseada em hierarquia, necessitando de conhecimento sobre a topologia da rede para estruturar a comunicação entre nodos [1]. O DAG enquadra-se nesta classe de algoritmos visto basear-se na criação de um grafo orientado acíclico sobre a topologia da rede. Apesar deste requisito extra acrescentar algum custo neste esquema, assim que o grafo fica formado (a nossa estrutura de routing), o problema de agregação tem por base um domínio já bastante estudado, Grafos. Passamos então a abordar o problema e agregação com uma estrutura matemática bastante familiar. O DAG em si traz uma abordagem nova no sentido em que permite a cada nodo ter mais que um nodo pai. Esta escolha despertou a nossa atenção visto ser uma forma simples de combater um problema muito comum neste tipo de algoritmos: falhas de rede.

4 Descrição do Simulador

4.1 Implementação

O primeiro passo do simulador é a geração de um DAG sobre o qual será executado o algoritmo. Da nossa perspectiva, a melhor maneira de conseguir construir um DAG válido será, através da biblioteca Networkx de Python gerar um grafo Erdős-Rényi através do método gnp_random_graph, ao qual mantemos as ligações que apontam de nodos com id inferior para nodos com id superior. No final disto, é ainda necessário garantir que o grafo gerado é "Connected"e contém o número de nodos pedidos.

O simulador, agindo de acordo com a especificação do algoritmo, faz broadcast a partir do root_node (nodo com id superior, a partir do qual conseguimos chegar a todos os outros nodos), de uma "Request Message"que contém a operação a ser executada e os nodos pais do nodo que a enviou. À medida que os vários nodos do grafo vão recebendo esta mensagem, actualizam na sua estrutura os seus pais, a partir do remetente das mensagens que recebem, e os seus avós, que vêm no payload dessas mensagens.

Em seguida, caso a operação seja passível de ser particionada em elementos mais pequenos, é criada uma nova "Request Message"com cada um desses elementos e broadcasted para o sub-grafo desse nodo, e o processo repete-se.

Caso não seja possível particionar mais a operação, por exemplo, o vector par somar apenas contém 2 elementos, o nodo executa a operação, e envia o resultado para ser agregado. Este resultado é marcado para ser entregue no avô preferido, o qual o nodo actual tenha mais caminhos (pais) por onde ir, e envia essa mensagem a esses pais, que quando a recebem vêm que não é destinada a eles e fazem "forward". O nodo destino, agrega todas as mensagens com ids únicos para chegar ao resultado parcial.

4.2 Avaliação e análise de desempenho

Para permitir avaliar o desempenho do algoritmo testado, o simulador tem algumas capacidades de replicar condições a que um sistema está exposto, como por exemplo, falhas em algum componente do sistema ou na rede que impliquem perdas de mensagens, sendo possível escolher uma percentagem de mensagens transmitidas a serem perdidas.

Para além disso, é ainda possível escolher o número de execuções que o simulador faz sobre determinadas condições para poder tirar resultados conclusivos fidedignos, alterando tal como anteriormente referido, por exemplo a percentagem de mensagens perdidas ou até o número de nodos do sistema.

É possível implementar qualquer operação, desde que obedeça a uma interface específica, para testar o simulador, neste momento, estão, por exemplo, implementadas as funções SUM e COUNT.

Recorreu-se ainda à biblioteca matplotlib para geração de alguns gráficos que permitam ver e interpretar os resultados obtidos da simulação dos algoritmos, alguns dos quais apresentados e analisados em seguida.

5 Desempenho do algoritmo

5.1 Variação dos erros em relação ao aumento substancial de mensagens perdidas

Neste primeiro gráfico, podemos observar alguns resultados da simulação do cálculo distribuído da soma de valores de um vector no sistema simples com um algoritmo de comunicação em árvore simples, em que cada nodo apenas tem um pai e apenas um ligação a esse mesmo pai.

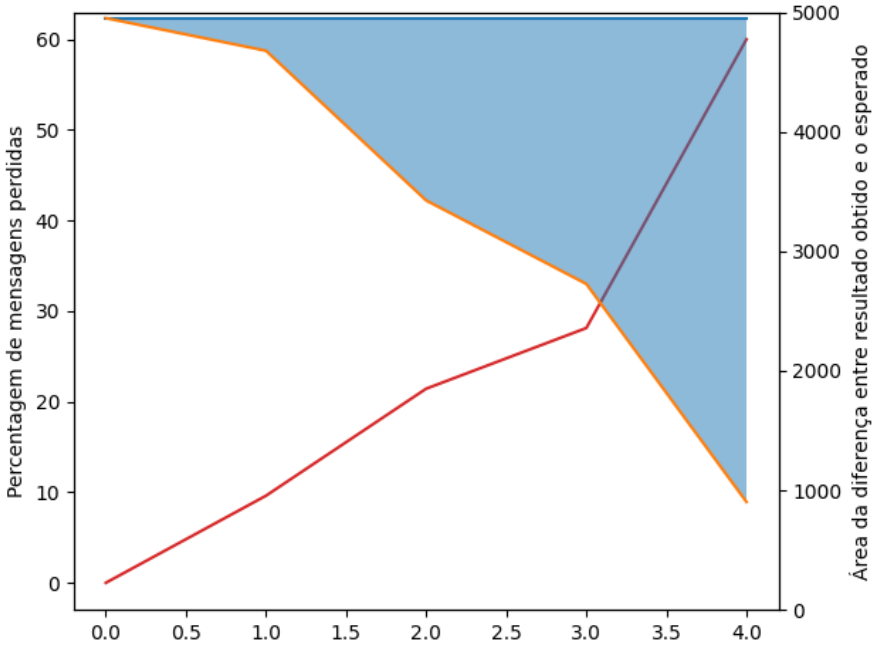

Figura 1: Algoritmo Simple Tree

É indiscutível que mal se começa a injectar falhas que causem perdas de mensagens no sistema o algoritmo muito rapidamente começa a dar valores completamente errados (+/- a partir de 20% de perdas).

No gráfico seguinte, podemos facilmente analisar a robustez do algoritmo DAG em termos da qualidade da rede, ou seja, a sua capacidade de manter resultados correctos independentemente de várias falhas, que implicam perdas de até 50% das mensagens, mantendo quase até ao final uma boa performance em termos de precisão dos resultados finais.

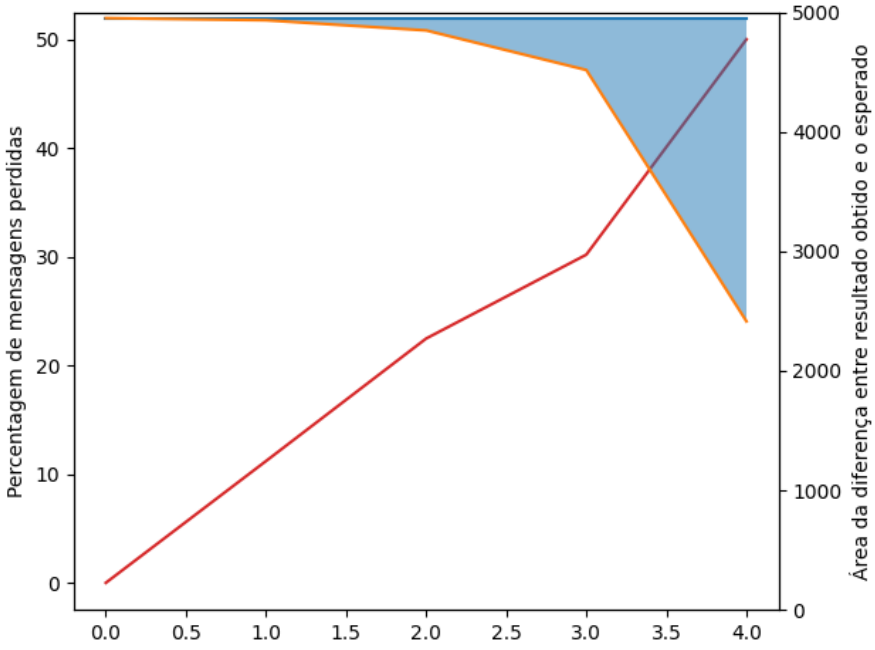

Figura 2: Algoritmo DAG

5.2 Diferença de resultados em consequência da variação de falhas

No seguinte gráfico, podemos observar, para um total de 20 iterações do simulador, com 14 nodos e um probabilidade de perda de mensagens de 10%, a azul a diferença (*500) entre o resultado final agregado e o resultado esperado, e a verde a percentagem real de mensagens perdidas nessa iteração do simulador:

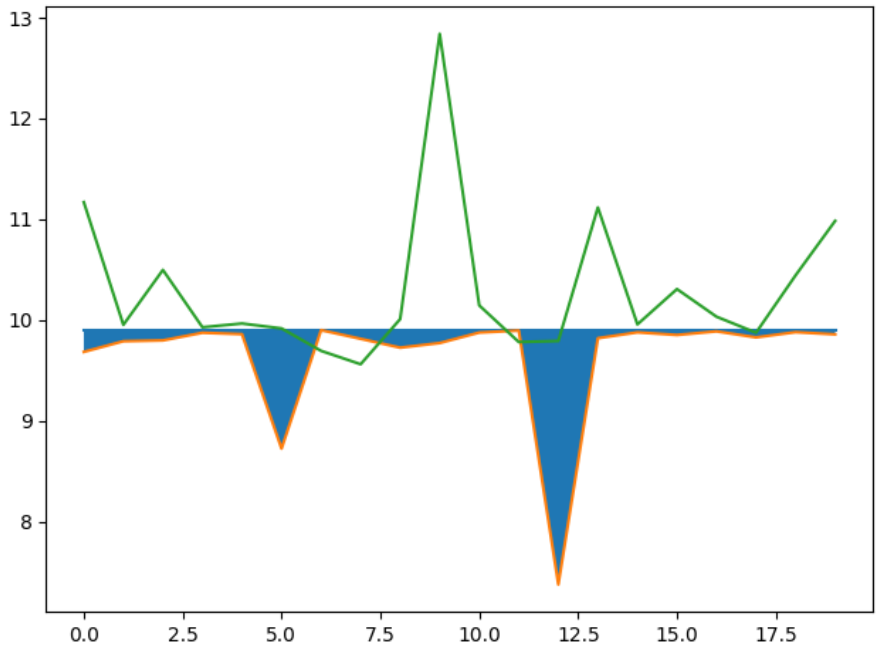

Figura 3: Diferença entre resultado esperado (*500) e obtido e percentagem de mensagens perdidas

Neste gráfico, fica claro que pequenas variações da percentagem de mensagens perdidas não é o maior factor de impacto no resultado final, apesar de a falha ser a causa destes erros, a qualidade de construção do grafo e consequentes caminhos alternativos entre nodos é que definem se existe tolerância a falhas e, por isso, robustez do serviço.

5.3 Variação dos erros em relação ao aumento dos nodos que compõe o sistema

Fazendo a média de 100 execuções de simulação para cada quantidade de nodos, podemos construir o seguinte gráfico:

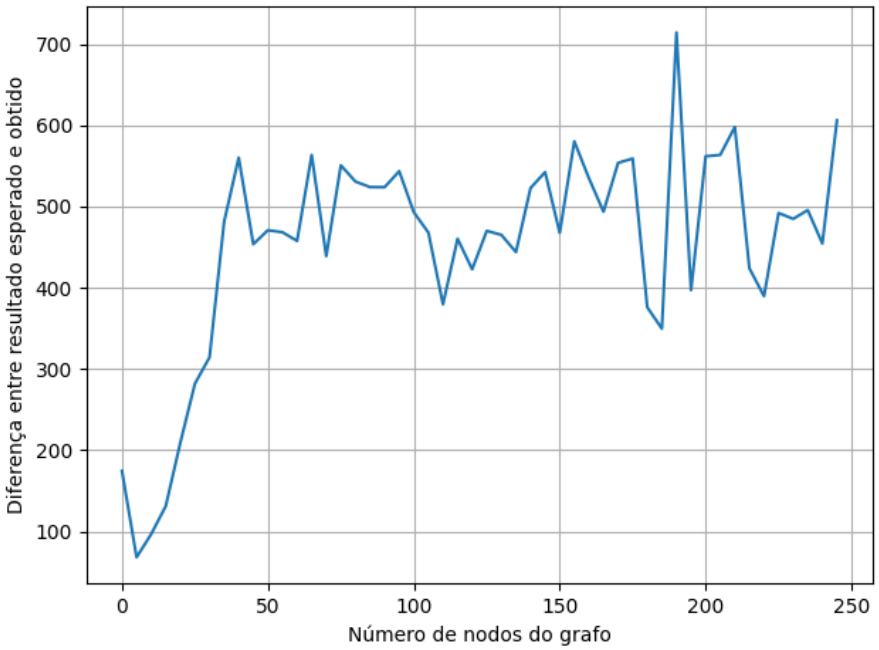

Figura 4: Evolução do erro de resultados em relação ao aumento do número de nodos do grafo

Surpreendentemente podemos verificar que, inicialmente, à medida que aumentamos o número de nodo o erro de cálculo aumenta exponencialmente (entre os 2 e os 45 nodos), estabilizando por completo em seguida.

6 Avaliação do algoritmo com recurso ao simulador

Apesar das óbvias vantagens e claros efeitos positivos na melhoria na robustez do serviço no que diz respeito à tolerância a falhas, quando comparado com outros algoritmos de topologia em árvore, estas melhorias são claramente insuficientes para obter resultados fidedignos e ainda com a desvantagem de sobrecarregar a rede com o envio de muitas mensagens com alguma probabilidade de serem descartadas.

Ainda assim, para além disto, o DAG tem uma excelente capacidade de se ajustar a mudança de views, uma vez que a descoberta de caminhos de filhos para avós, através de pais, é feita dinâmicamente à medida que se enviam mensagens "Request"pelo grafo. Mas é importante notar, que esta metodologia acarreta um ainda maior aumento no custo em termo do tamanho de mensagens enviadas.

Apesar de tudo, um sistema baseado neste algoritmo tem muitas potencialidades caso seja depois combinado com outras técnicas para descoberta e reenvio de mensagens perdidas, mas é impossível à priori prever se será ou não uma solução viável, porque se tivermos maneiras de detectar e reenviar mensagens poderá não valer a pena a imensa sobrecarga que estamos a causar à rede.

7 Melhoria na geração de DAG com garantia de propriedades

7.1 Matriz estática

Em vez de se gerar um grafo aleatório como anteriormente, ao qual depois eram removidas as ligações a mais, neste algoritmo de geração construímos todo a grafo manualmente.

Uma primeira melhoria foi garantir que todos os nodos têm sempre mais do que uma via de comunicação para os seus pais. Para isto, cria-se um matriz triangular superior, no qual preenchemos com 1 as ligações entre os nodos, neste caso, cada nodo está connectado aos dois nodos com ids imediatamente anteriores, à exceção do nodo 1 que apenas só pode ter uma ligação, porque só existe um nodo anterior.

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

que dá origem ao grafo

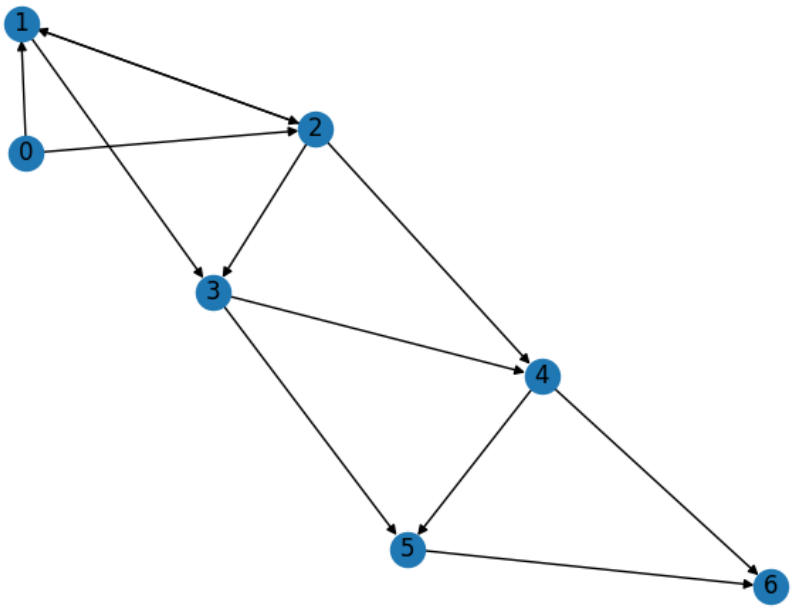

Figura 5: DAG

Assim, para condições de rede em que a perda de mensagens seja na ordem dos 10%, a probabilidade P(i) de perda de mensagens enviados por um nodo i é:

$$\mathrm{P}(1) = 0.1 \tag{2}$$

$$\mathrm{P(i)} = 0.01, \forall \mathrm{i} \in [2, 3, 4, 5, 6] \tag{3}$$

Caso a percentagem de mensagens perdidas seja 50%:

$$\mathrm{P}(1) = 0.5 \tag{4}$$

$$\mathrm{P(i)} = 0.25, \forall \mathrm{i} \in [2, 3, 4, 5, 6] \tag{5}$$

Um valor muito mais elevado do que seria desejável.

7.2 Matriz dinâmica

Mas este metodologia abre-nos a possibilidade de fortalecer o grafo resultante dependendo das condições de rede que prevemos ter, podendo ser adicionadas mais ligações entre grafos. Assim a matriz pode ser construída de acordo a probabilidade de perder mensagens. Obviamente acarretando sempre consigo um aumento muito grande na quantidade mensagens trasmitidas.

Para isso e recorrendo ao seguinte método para calcular o número de ligações:

```
shift = math.ceil(number_of_nodes/5) + round(network_lose_percentage / 10)
```

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

que dá origem ao grafo:

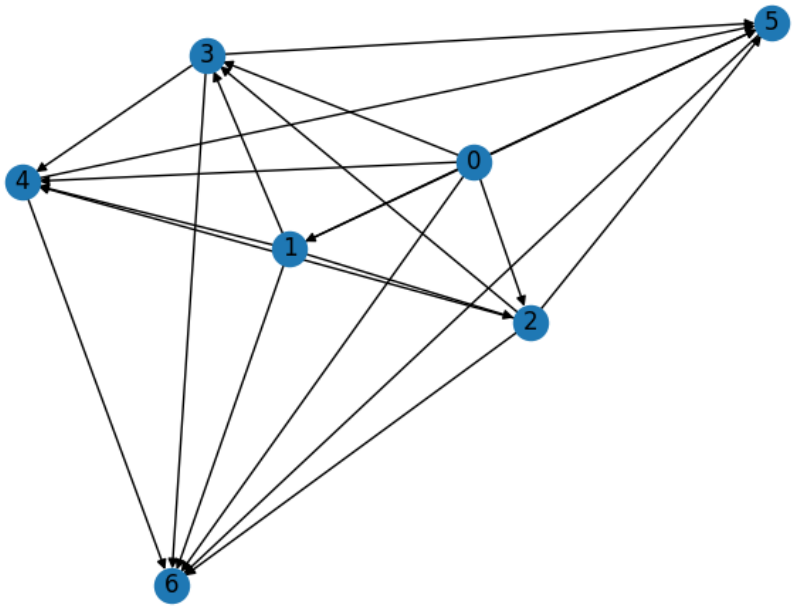

Figura 6: DAG

no qual:

P(1) = 0.5, P(2) = 0.25, P(3) = 0.125
P(4) = 0.0625, P(5) = 0.03125, P(6) = 0.015625

Mantendo o máximo de valores possíveis abaixo do 1% sem nunca deixa de ter um grafo acíclico.

7.3 Desempenho no simulador

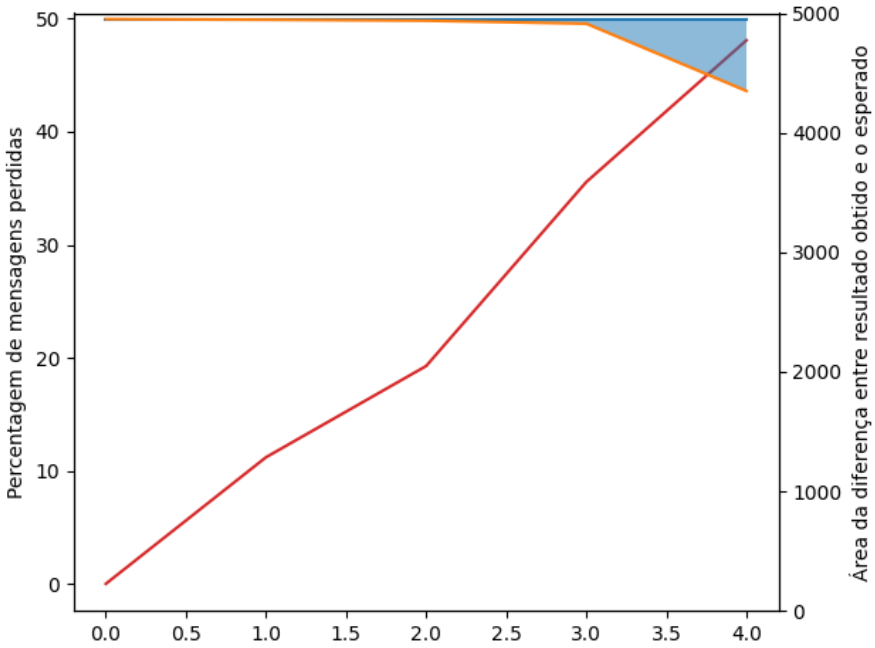

Figura 7: Diferença entre resultado esperado e obtido

Quando comparado com o gráfico 2, vemos uma gigante melhoria em termos de resistência às condições de rede, mantendo os valores quase 100% corretos até 35% de perdas de mensagens, e mesmo depois a diferença de resultados obtido e esperados é muito baixa.

8 Conclusão

A capacidade de testar algoritmos num simulador deste género é, sem dúvida, essencial para perceber e analisá-los. Como mostrado ao longo deste relatório, foi possível, de forma segura, eficiente e rápida, simular um grande conjunto de implementações reais deste género de algoritmos num ambiente controlado e de baixo custo.

Esta facilidade de teste de um alargado número de conjunturas, que aliam, entre outras, condições de rede melhores ou piores, um conjunto facilmente modificável de nodos, quer em termos de quantidade, quer em termos de conexões entre eles, permite testar e avaliar caso a caso, e dependendo do objetivo final, cada um dos algoritmos considerados.

Neste caso específico, através do simulador, conseguimos de maneira intuitiva verificar os resultados muito superiores de uma implementação DAG quando comparada com uma "tree"normal,

mas que mesmo assim, dá origem a algumas falhas, embora inferiores, sendo instantâneo perceber que este algoritmo não pode ser usado para situações onde seja essencial o rigor dos resultados finais.

Graças ao simulador, é também fácil analisar a evolução do comportamento destes algoritmos de acordo com mudanças ao nível de falhas de rede ou de aumento substancial do número de nodos do sistema, por exemplo.

9 Bibliografia

Referências

[1] A Survey of Distributed Data Aggregation Algorithms
Paulo Jesus, Carlos Baquero, Paulo Sérgio Almeida. Technical Report (September, 2011)

[2] DAG based In-Network Aggregation for Sensor Network Monitoring
S Motegi, K Yoshihara, and H Horiuchi. International Symposium on Applications and the Internet (SAINT), page 8 (2005)