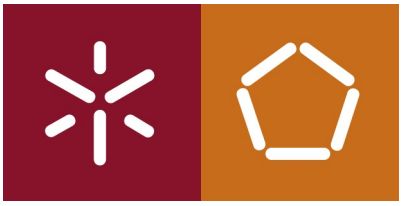

Universidade do Minho
mestrado integrado em engenharia informática

Sistemas Distribuídos em Larga Escala

(2º Semestre / 4º Ano)

Perfil de Mestrado: Sistemas Distribuídos

Relatório do Trabalho Prático

Decentralized Aggregation - Push-Sum

Grupo 10:
Henrique Pereira (a80261)
João Silva (a82005)
Miguel Carvalho (a81909)

Conteúdo

1 Introdução

No âmbito da Unidade Curricular de Sistemas Distribuídos em Larga Escala, incluída no perfil de Sistemas Distribuídos do Mestrado Integrado em Engenharia Informática, foi-nos proposta durante as aulas práticas a implementação de um simulador de eventos discretos para testar algoritmos distribuídos. Ora, o trabalho prático sobre o qual este relatório incide consiste na utilização desse simulador para estudar, implementar e testar um algoritmo distribuído de agregação.

Como referido no artigo fornecido pelos docentes - *"A Survey of Distributed Data Aggregation Algorithms"* [1] -, a agregação de dados é um dos blocos fundamentais na construção de sistemas distribuídos modernos, permitindo a descoberta de informações globais de um sistema que poderão ser utilizadas depois em outros sistemas ou algoritmos distribuídos. Exemplos de agregações são o tamanho de uma rede ou o espaço disponível em disco num sistema de ficheiros partilhado. Assim, a agregação pode ser definida como a "habilidade de sumariar informação"[1].

Os algoritmos de agregação apresentados no artigo disponibilizado são de diferente natureza, quer a nível de comunicação, quer a nível de computação, tendo cada um deles as suas vantagens e as suas desvantagens.

Nas secções seguintes iremos indicar e justificar a escolha do algoritmo, bem como explicitar a nossa interpretação do mesmo utilizando o simulador por nós desenvolvido.

2 Algoritmo

O algoritmo por nós escolhido incialmente foi o *Hybrid Protocol* [2], desenvolvido no ano de 2008. Este pareceu-nos muito interessante pelo facto de tentar juntar o melhor de dois mundos: os algoritmos com comunicação hierárquica (que são mais eficientes, mas não são tolerantes a faltas) e com comunicação baseada em *gossip* (que são tolerantes a faltas, mas necessitam de muitas mensagens para chegar a um resultado, piorando o seu desempenho). O algoritmo resulta de uma adaptação dos algoritmos *TAG* [3] (hierárquico) e *Push-Sum* [4] (*gossip*). Porém, ao iniciarmos a sua adaptação para este trabalho prático, começamos pelo desenvolvimento do algoritmo *Push-Sum* e percebemos desde logo que havia muito para explorar, pelo que optamos por selecionar apenas este. Assim, as secções que se seguem são dirigidas ao algoritmo Push-Sum [4].

2.1 Breve descrição

Como foi referido, o algoritmo *Push-Sum* [4] é um algoritmo baseado em *Gossip*, utilizado para computar funções de agregação tais como somas e médias.

O algoritmo funciona através da distribuição de um par de valores por toda a rede de nodos, ou seja, por rondas, de dado em dado tempo, cada nodo propaga pelos vizinhos dois valores: s_{ti} e w_{ti}, onde s_{ti} é a soma dos valores trocados e w_{ti} diz respeito ao peso associado a esta soma, ambos no nodo i e na ronda t.

Tendo isto em conta, para calcular diferentes funções de agregação basta alterar os valores iniciais de cada nodo, ou seja, para uma média, tendo em conta que cada nodo tem inicialmente como s o seu valor, o peso de todos inicialmente será 1. Por outro lado, para uma soma, mantendo inicialmente os seus valores originais, os seus pesos passarão para 0, à exceção do nodo inicial que terá como peso o valor de 1.

O método de funcionamento do algoritmo é o seguinte: em cada iteração (ou ronda), um vizinho é escolhido de forma aleatória e metade dos seus valores atuais são enviados para o vizinho selecionado e a outra metade fica para o próprio nodo. O nodo vizinho, ao receber os valores, soma-os aos seus próprios. Uma estimativa de agregação pode ser computada em cada nodo em qualquer ronda, dividindo o valor da soma pelo peso. A precisão da agregação tende a aumentar com o número de rondas, convergindo eventualmente no valor correto.

Segundo os investigadores por trás deste algoritmo, este assenta na propriedade fundamental de conservação de massa, ou seja, a soma de todos os valores da rede (quer locais, quer em trânsito) deve permanecer constante ao longo de toda a execução. Assim, torna-se fundamental a existência de um mecanismo que detete faltas, isto é, que saiba quando uma mensagem não chegou ao destino. Quando isto

sucede, o nodo repõe a massa enviando a mensagem em falta novamente, mas desta vez para o próprio nodo.

2.2 Motivos de escolha

Para a seleção do algoritmo seguimos o *survey "A Survey of Distributed Data Aggregation Algorithms"* [1] fornecido e tivemos em conta vários aspetos que serão abordados nas subseções abaixo.

2.2.1 Comunicação

O primeiro aspeto que tivemos em consideração para a escolha do algoritmo foi a **comunicação** que contempla tanto o protocolo de *routing* a considerar, como a topologia da rede em que este protocolo opera.

Como o *survey* indica, podemos dividir os algoritmos de agregação em 3 categorias: estruturados (baseados em hierarquia), não estruturados (baseados em *gossip* ou *flooding*) e os híbridos.

Os estruturados apresentam excelentes resultados, porém não são tolerantes a faltas. Como a sua estrutura é em árvore, quando um nodo falha, os resultados serão fortemente influenciados por esta falha. Os não estruturados, ao contrário dos anteriores, apresentam resultados muito superiores quando existe maior probabilidade de falta, mas são menos eficientes, pois exigem uma troca de mensagens muito maior.

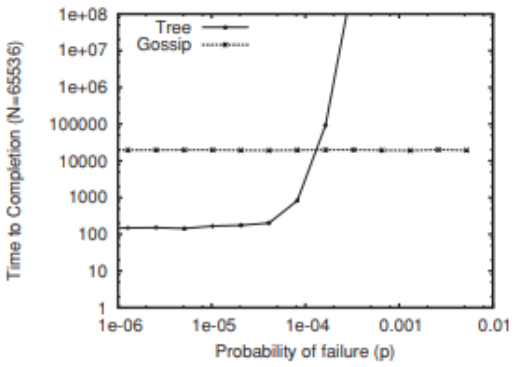

Figura 1: Comparação entre Algoritmos Estruturados e Não Estruturados

Tendo isto em atenção, decidimos que um algoritmo baseado em *gossip* seria mais interessante de abordar. Como referimos anteriormente, ponderamos em primeiro lugar escolher o algoritmo *Hybrid Protocol* [2] por tentar retirar os melhores aspetos de cada tipo, mas o *Push-Sum* [4] (no qual se baseia o anterior) também nos pareceu muito apelativo a partir do momento que começamos a trabalhar com ele.

2.2.2 Computação

O segundo aspeto em que baseamos a nossa decisão foi a maneira como o algoritmo realiza a **computação** para obter os resultados de agregação.

Tal como na parte da comunicação, podemos dividir os algoritmos em vários tipos consoante a sua forma de computação: hierárquicos (que também são mais eficientes, mas não tolerantes a faltas), de *averaging* (que toleram faltas, mas necessitam de mais mensagens, convergindo num resultado com o passar do tempo), *sketches* (que são mais fiáveis que os hierárquicos, adicionando redundâncias, mas que trazem erros de aproximação), *digests* (que reduzem todos os inputs a uma estrutura de tamanho fixo usando métodos probabilísticos e perdendo alguma informação, caindo novamente em erros de aproximação) e *sampling*, ou seja amostragem (onde são selecionados apenas alguns nodos para a computação, podendo não se obter o resultado real, mas é mais eficiente no que toca à troca de mensagens).

Tendo isto em mente, achamos que os algoritmos de *averaging* seriam os mais interessantes, pois eventualmente convergem num resultado correto, não estando susceptíveis a faltas, apesar de necessitarem de uma maior troca de mensagens.

Como o Push-Sum é um algoritmo baseado em *gossip* e *averaging*, achamos que este seria um bom algoritmo para implementarmos no nosso simulador. Esta implementação será descrita nas secções que se seguem.

2.3 Modelo do Sistema

Dado o problema da computação de agregações descentralizadas, mais especificamente do algoritmo *Push-Sum* [4], definimos o modelo em que o sistema irá funcionar. Assim sendo, a partir deste momento iremos assumir um modelo de sistema onde:

1. Existe um conjunto de nodos com identificadores sequenciais;
2. Nodos estão conectados numa dada topologia de rede, e conhecem apenas os identificadores dos seus vizinhos;
3. Qualquer que seja a topologia da rede usada, esta terá que se manter estática, não suportando faltas nos nodos, nem faltas do tipo *Crash-Link*;
4. Nodos comunicam através de passagem de mensagens, podendo os seus canais experienciar faltas omissivas do tipo *Receive, Send e General*;
5. O sistema é totalmente assíncrono, sendo possível adicionar algum tipo de limite temporal na latência (i.e. Timeouts);

2.4 Implementação do Algoritmo

Dado o problema descrito acima e o modelo assumido, a solução que construímos tem algumas particularidades descritas a seguir.

2.4.1 Modelo Assíncrono

Dada a natureza do modelo escolhido (assíncrono), deixamos de parte a solução deste problema com rondas síncronas, e passámos a fazê-lo com iterações assíncronas onde os nodos não se encontram sincronizados no que toca à ronda.

Uma ronda assíncrona neste contexto é o processo dum nodo enviar uma mensagem de *Request* para um conjunto aleatório de vizinhos e esperar receber as suas respostas. Após receber as respostas de todos os vizinhos, poderá dar início a uma nova ronda. Destas forma os nodos não estão sincronizados nas rondas em que se encontram.

2.4.2 Deteção de Perdas de Mensagens

Devido à propriedade de conservação de massa em que este algoritmo se baseia, a perda de mensagens arruinaria a convergência do mesmo para um resultado correto.

Deste modo, para garantir a convergência para o resultado correto precisamos de garantir a entrega de todas as mensagens entre os nós. Para isso, decidimos implementar um sistema baseado no TCP (*Transmission Control Protocol*).

O TCP assegura a entrega das suas mensagens à custa de duas componentes do seu protocolo: o ACK e o RTO.

O ACK é o tipo de mensagem que um recetor envia ao remetente para que este tenha a garantia que a sua mensagem foi entregue. Isto é importante pois o remetente irá reenviar todas as mensagens que não forem respondidas com um ACK (para que estas não se percam).

No entanto, não pode proceder ao reenvio de uma mensagem logo após ao envio - necessita de aguardar um tempo à espera do ACK a ser enviado pelo recetor. Esse tempo de espera denomina-se RTO (*Retransmission Timeout*) e é calculado dinamicamente ao longo da execução da simulação.

O RTO é definido por cada par remetente-recetor e é pesado consoante o tempo medido entre o envio de uma mensagem e a receção do ACK correspondente à mesma. Desse modo, minimizam-se as mensagens reenviadas cujo ACK não se perdeu mas estava atrasado, aumentando assim a eficiência da retransmissão.

Essas poucas mensagens que são reenviadas antes da receção do ACK, irão chegar ao recetor replicadas. Ora, para garantir a conservação de massa, o recetor não pode tomar os valores das mensagens replicadas. Com isso em mente, o recetor verifica a ronda de cada mensagem pois sabe que o remetente não pode enviar

várias mensagens do mesmo tipo - *Request* ou *Response* - na mesma ronda para o mesmo destinatário. Quando deteta um caso destes, o recetor ignora as mensagens replicadas e aproveita apenas os valores da primeira mensagem que recebeu.

2.4.3 Node-Crash e Crash-Link

Dada a natureza do algoritmo, faltas deste tipo não podem ocorrer, uma vez que poderíamos estar a quebrar a propriedade de *conservação de massa*.

No caso dum **Node-Crash**, o nodo deixaria de processar pelo que não contribuiria para o avanço do algoritmo e levaria a resultados incorretos. Uma forma de mitigar este problema seria um nodo antes de falhar entregar o seu par de valores a um nodo vizinho, mas esta é uma execução muito otimista e provavelmente pouco realista, mas desta forma levaria a uma correta agregação no final.

No caso dum **Crash-Link** a propriedade de *conservação de massa* pode ser severamente afetada. Imagine-se o caso em que o nodo A envia uma mensagem para o nodo B, entretanto ocorre um *crash-link*. O nodo A não sabe se a sua mensagem foi entregue em B, logo não poderá tomar a decisão se repõe a massa perdida ou não. O mesmo ocorre do lado do nodo B que no caso de receber a mensagem tentará responder mas não conseguirá, pelo que não sabe se mantém o par recebido ou o descarta pois não tem conhecimento sobre a decisão do nodo A.

2.4.4 Deteção de Terminação

Na descrição do algoritmo *Push-Sum* [4], os autores não deixam qualquer noção de como fazer a terminação do algoritmo, no entanto é útil ter algum critério de terminação. Poderia ser o simulador a parar o algoritmo quando este convergisse, mas este cenário parece pouco realista pelo que definimos um critério de paragem nos nodos. Esta decisão levanta algumas questões, por exemplo, o nodo deve parar quando a aproximação for boa o suficiente, mas não conhecendo a topologia da rede, como se sabe que a aproximação é de facto boa?

Para construir um critério de paragem inspiramo-nos no algoritmo 2 do artigo *"Extreme Propagation: Fast Distributed Estimation of Sums and Network Sizes"* [5], que dita que o algoritmo deve parar quando não se recebem novos *updates* dos vizinhos durante um número de rondas seguidas. Ora no nosso caso como fazemos *multicast* para um conjunto aleatório de vizinhos em cada ronda, fica complicado avaliar esta propriedade uma vez que em cada ronda receberíamos informação de nodos diferentes.

Desta forma, adaptamos este critério ao nosso caso, onde em vez do nodo avaliar as "news"provenientes dos vizinhos, avalia o valor da sua agregação. Quando o valor da sua agregação se mantiver igual durante X rondas (valor customizável) então esse nodo pára de produzir novas rondas, mas continua a responder a pedidos

dos seus vizinhos e continua a avaliar a sua agregação. Se em algum momento um nodo que esteja em estado de terminação receber uma mensagem que altere o valor da sua agregação para um valor diferente dos anteriormente observados, então este nodo sai do estado de terminação e despoleta uma nova ronda. Desta forma, ao fim de algum tempo o algoritmo pára, tendo cada nodo o valor da agregação final.

3 Simulador

3.1 Breve descrição

O simulador, como o nome indica, tem como propósito simular eventos que acontecem numa rede de nodos e observar os efeitos dos mesmos num ambiente controlado. No nosso caso em concreto, pretendemos observar a evolução do algoritmo *Push-Sum* [4] e a convergência do mesmo em redes distintas.

Para o efeito, o simulador faz recurso de um conjunto de nodos capazes de responder a mensagens, das distâncias entre os mesmos no grafo definido e de um relógio lógico para garantir a ordenação dos eventos.

No entanto, este simulador tem a particularidade de permitir apenas o envio de mensagens como evento.

3.2 Envio de Mensagens

O envio de mensagens no simulador é tratado com recurso a uma lista de eventos pendentes, dos quais em cada iteração o simulador seleciona o que ocorre no instante temporal mais baixo.

Selecionado o evento, o nodo recetor descrito no mesmo é invocado para tratar a mensagem contida no evento. Ao tratar a mensagem, o nodo devolverá novos eventos que são adicionados à lista de eventos pendentes, de modo a que o simulador os trate mais tarde.

Ao adicionar estes eventos à lista dos pendentes, o simulador calcula o instante temporal em que cada um deverá ocorrer com recurso ao instante em que a mensagem foi processada, aos nodos envolvidos na mensagem e ao atraso indicado pelo nó.

3.3 Agendamento de eventos locais

Para o agendamento de eventos locais, o simulador, por si só, não possui qualquer funcionalidade especializada. No entanto, possibilita que um nó indique um atraso temporal em cada mensagem que retorna, de modo a que esta seja entregue apenas ultrapassado esse atraso.

Ora, este atraso pode ser usado por cada nó para o agendamento de eventos locais. Para o efeito, o nó que pretende que um evento local ocorra após um *timeout* T, deve devolver ao servidor uma mensagem com atraso T e com ele mesmo como destinatário.

O simulador, ao receber esse evento, fá-lo-á decorrer precisamente após T unidades temporais, já que a distância entre um nó e ele mesmo é, naturalmente, 0.

3.4 Injeção de faltas

A injeção de faltas aleatórias no simulador é feita através de uma probabilidade de falta, indicada na sua inicialização. A cada iteração da simulação, o simulador gera um número aleatório e caso esse esteja abaixo da probabilidade de falta, o simulador ignora o evento que devia tratar na iteração em questão.

No entanto, nem todos os eventos podem ser ignorados na simulação. Como o simulador apenas permite o envio de mensagens entre nodos, os eventos locais ou de inicialização da simulação devem ser protegidos e não tratados como eventos normais. Desse modo, estes eventos são sempre executados, podendo apenas os eventos que envolvem nós distintos pertencentes à rede ser ignorados.

4 Benchmark

Para avaliar o algoritmo implementado realizamos alguns testes ao mesmo, modificando várias configurações que influenciam os resultados finais observados. Essencialmente focamos a nossa análise na complexidade **temporal** (em milissegundos) de **mensagens**.

4.1 Configuração inicial

Os valores que são possíveis variar e que têm interferência nos resultados finais são, o **Fanout**, a **Percentagem de Erro**, o tamanho da lista **No News** e a **Topologia** da rede.

O **Fanout** é o número de vizinhos para que é enviada uma mensagem numa determinada ronda. A **Percentagem de Erro** dita a probabilidade duma mensagem se perder. O **No News** é o valor que define a terminação do algoritmo, pois quando um nodo observar o valor da sua agregação *No News* vezes seguidas sem se alterar então sabe que pode terminar. A **Topologia** da rede é a forma como os nós se organizam consoante as suas ligações.

Para realizar os primeiros testes, então fixamos os seguintes valores:

- Fanout = 1
- Percentagem de Erro = 0%
- No News = 5
- Topologia = nenhuma inicialmente

4.2 Topologia

Neste teste escolhemos a topologia a usar na análise dos restantes parâmetros falados na secção anterior. Para isso, fizemos testes a 3 topologias, **Erdos-Renyi**, **Barabasi-Albert** e **Watts-Strogatz**. Os resultados obtidos apresentam-se de seguida.

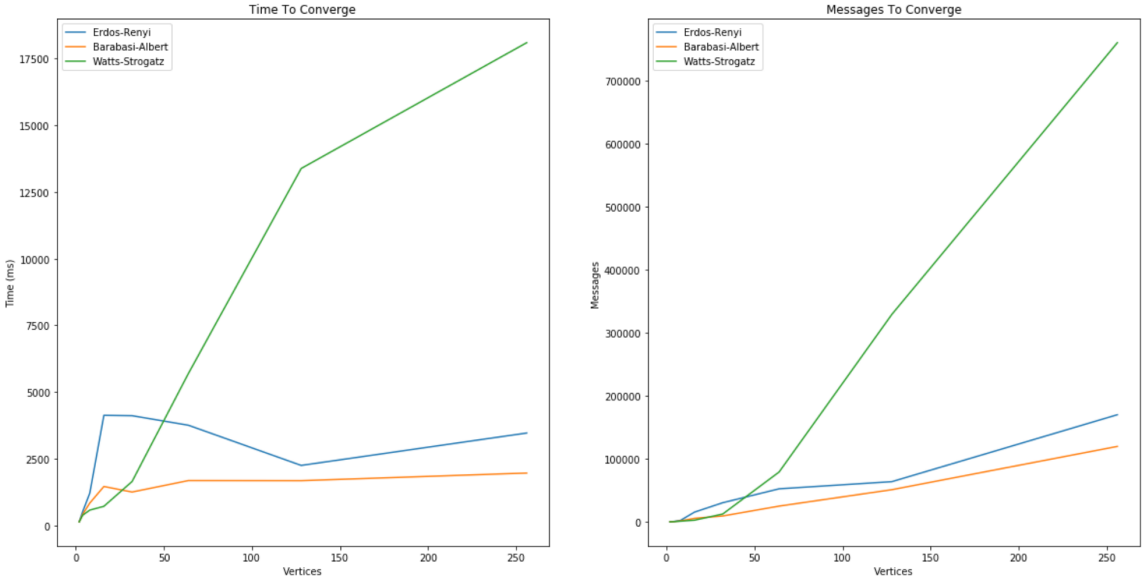

Figura 2: Comparação entre Topologias

Dados os resultados obtidos decidimos continuar a realizar os testes numa topologia **Barabasi-Albert**, uma vez que esta apresentou resultados melhores no que toca à complexidade temporal e de mensagens.

4.3 Fanout

Nos testes anteriores usamos um **Fanout** igual a 1, pelo que em cada ronda o algoritmo apenas enviava uma mensagem para um vizinho direto e esperava a sua resposta. Esta obviamente não é a forma mais eficaz, mas também é necessário perceber que aumentar em demasia este número poderá ter um impacto negativo na complexidade das mensagens.

Variamos o número do **Fanout** entre 1 e 5, porque para mais de 5 a complexidade de mensagens pode já ser elevada. Os resultados são apresentados a seguir.

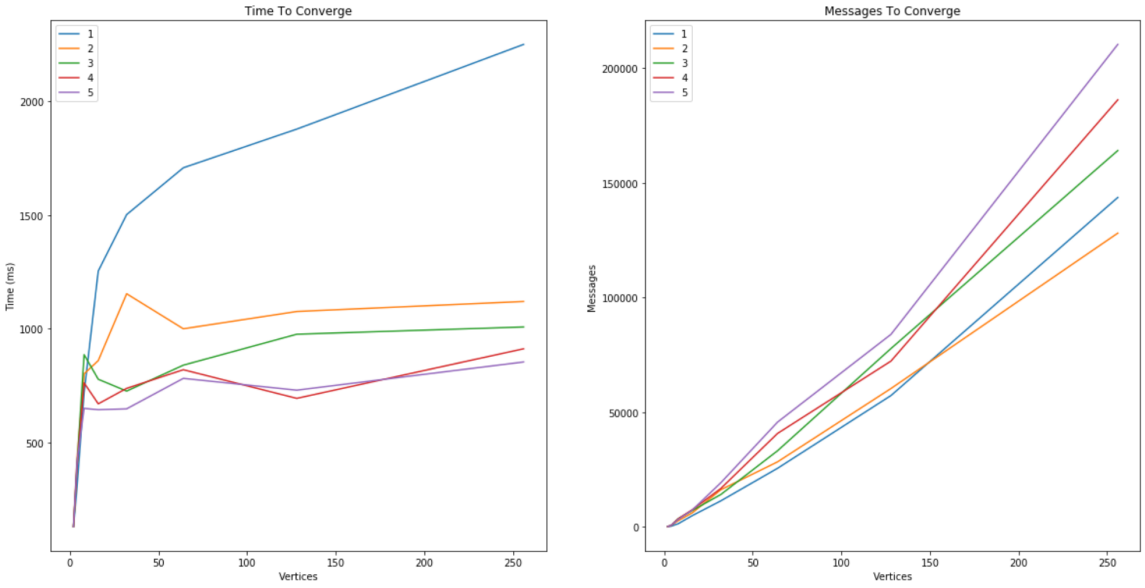

Figura 3: Comparação entre Fanouts

Tendo em conta os resultados obtidos, optamos por escolher um **Fanout** igual a **2**, pois apesar de não ter os piores resultados em comparação com os restantes *fanouts* na complexidade temporal, é o melhor no que toca à complexidade de mensagens.

4.4 Percentagem de Erro

Após avaliar os primeiros parâmetros, acrescentamos alguma percentagem de erros ao sistema, não só para mostrar que o algoritmo é tolerante a faltas mas também para perceber o impacto dos erros nas complexidades a avaliar.

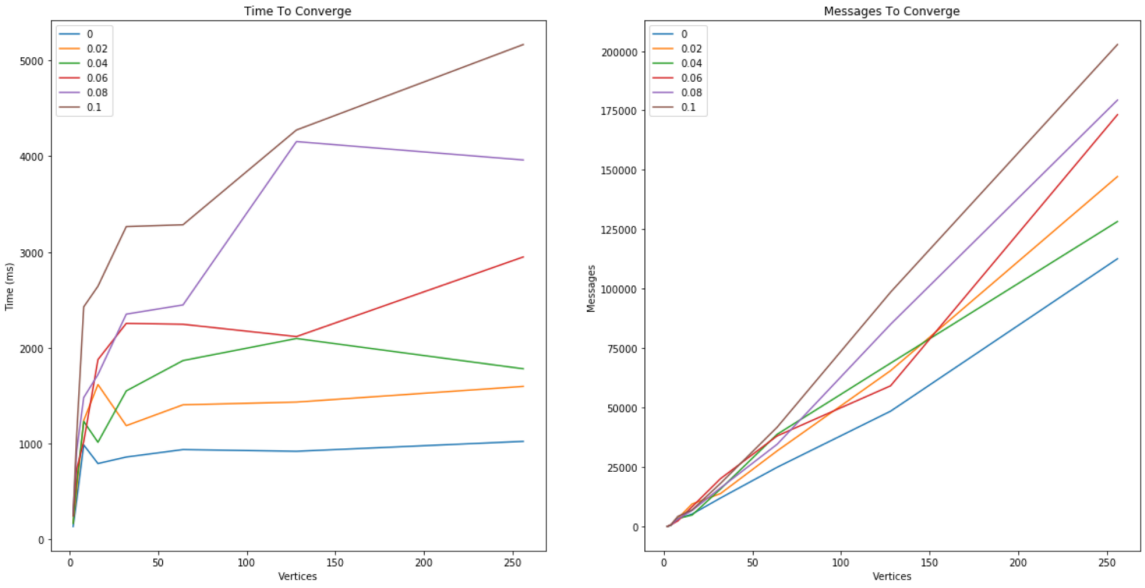

Figura 4: Comparação entre percentagens de erro

Como esperado, com o aumento da percentagem de erros, tanto o tempo necessário, como as mensagens necessárias para convergir aumentam no mesmo sentido. Para os testes seguintes escolhemos uma percentagem igual a **4%** uma vez que na complexidade de mensagens obteve-se melhores resultados para este valor.

4.5 No News

Este parâmetro, exige que se tenha alguma atenção nos valores escolhidos, pois sendo bastante otimista e escolhendo um número bastante pequeno pode acontecer que os nodos no final não tenham convergido. Sendo bastante conservador e escolhendo um valor alto, existe uma grande probabilidade de todos os nodos convergirem, no entanto serão necessárias mais rondas até que isso aconteça, afetando assim o tempo necessário de terminação do algoritmo.

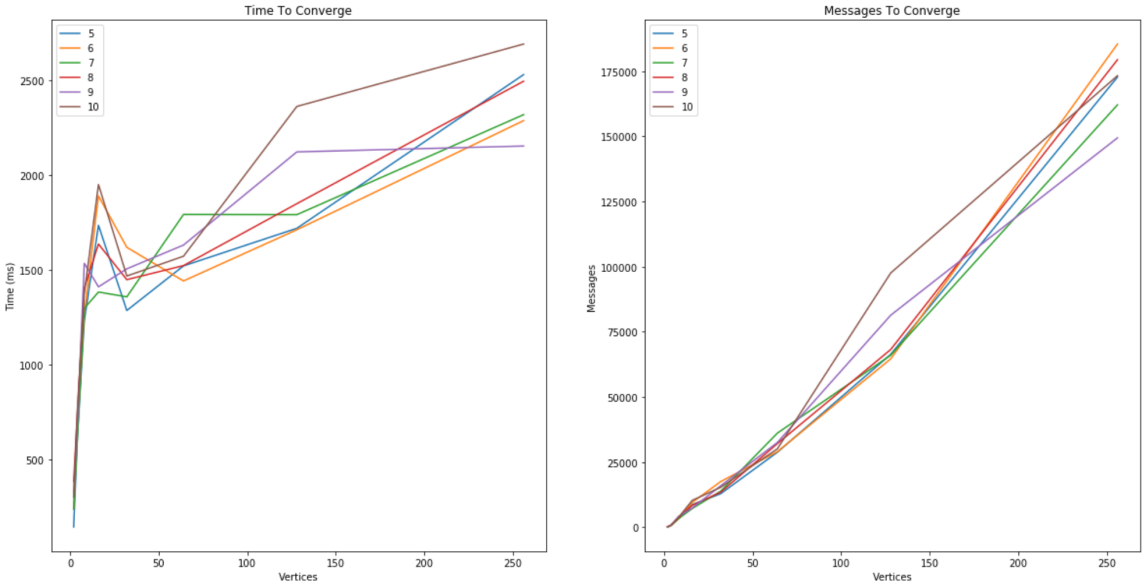

Figura 5: Comparação entre valores para No News

Desta forma, variamos este parâmetro de **5** a **10** e percebemos que não houve diferenças significativas nas complexidades observadas.

4.6 Resultados Finais

Tendo em conta os testes anteriormente desenvolvidos, para uma configuração com:

- uma topologia **Barabasi-Albert**;
- **256** nodos;
- um fanout de **2**;
- uma percentagem de erro de **4%**;
- um tamanho do array No News igual a **9**;

Obtivemos um tempo que ronda, em média, os **2 segundos** para que houvesse convergência, onde foi necessário, em média, cerca de **140 mil mensagens** trocadas entre os nodos.

5 Conclusão

Olhando em retrospetiva para o trabalho realizado, o simulador desenvolvido nas aulas práticas da UC de Sistemas Distribuídos em Larga Escala foi bastante útil pois simplificou o processo de implementação e teste do algoritmo em causa (Push-Sum). A única alteração necessária foi a remoção da *periodic view-change*, pois o algoritmo não a permitia.

Foi bastante interessante o desenvolvimento deste projeto, assim como o caminho que percorremos para o realizar: apesar de iniciarmos e termos decidido um algoritmo diferente do que aquele em que incide este relatório, ao iniciarmos a sua implementação decidimos explorar um dos algoritmos componentes do inicial, ou seja, o Push-Sum. Este trabalho permitiu-nos explorar *papers* científicos de autores internacionais, permitindo-nos compreender de forma mais extensiva acerca de alguns dos algoritmos de agregação existentes, bem como acerca dos vários tipos destes existentes.

Achamos que cumprimos os vários pontos requeridos pelos docentes, à exceção do facto de o presente relatório exceder o número de páginas previsto.

Bibliografia

[1] P. Jesus, C. Baquero, and P. Almeida, "A survey of distributed data aggregation algorithms," *Communications Surveys Tutorials, IEEE*, vol. 17, 10 2011.

[2] L. Chitnis, A. Dobra, and S. Ranka, "Aggregation methods for large-scale sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, 03 2008.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, p. 131–146, Dec. 2003.

[4] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pp. 482–491, 2003.

[5] C. Baquero, P. S. Almeida, R. Menezes, and P. Jesus, "Extrema propagation: Fast distributed estimation of sums and network sizes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 668–675, 2012.