Universidade do Minho

Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

Sistemas Distribuídos em Larga Escala

Push Sum Protocol

Luís Capa - A81960
Pedro Capa - A83170

1 Introdução

Este relatório pretende descrever os principais pontos do trabalho prático da *UC* Sistemas Distribuídos em Larga Escala. Neste trabalho teríamos que escolher e fazer a implementação de um algoritmo, entre um conjunto de algoritmos sugeridos do artigo [1].

O algoritmo que decidimos escolher foi o *Push-Sum Protocol*. Com este algoritmo é possível fazer cálculos de algumas funções de agregação tais como o **COUNT**, **SUM** e **AVERAGE**.

Primeiro, explicamos o motivo da escolha deste algoritmo. De seguida, fazemos uma pequena descrição do protocolo e da nossa implementação. Depois, descrevemos o simulador, que desenvolvemos nas aulas práticas e utilizamos para validar a nossa implementação e de que forma nos ajudou a ver alguns problemas na implementação. Por fim, iremos mostrar alguns resultados da aplicação do simulador ao algoritmo.

2 Decisão

Existiram diversos motivos para a escolha deste algoritmo em particular. Primeiro, este algoritmo funciona para qualquer tipo de topologia, logo a topologia não era um fator que precisasse especial atenção quando testássemos o algoritmo, apesar de ser interessante testar a resposta do algoritmo a topologias diferentes. Segundo, este algoritmo em termos da categoria de computação pertence à categoria *Average*, que tem o princípio de convergir para o resultado correto, que é um principio seguido por muitos algoritmos noutras áreas. Por fim, se assegurássemos um princípio deste algoritmo, era garantido que iriamos obter o resultado correto.

3 Descrição do algoritmo

3.1 Protocolo

O algoritmo **Push-Sum** permite o cálculo das funções **SUM**, **COUNT** e **AVERAGE**, através da atribuição de um par de valores *(s, w)* a todos os nodos da rede e dependendo dos valores atribuídos inicialmente é possível calcular uma destas funções de agregação. O s representa a soma dos valores trocados e o w o peso associado à soma.

De forma a que todos os nodos calculem o valor das funções da agregação, cada nodo num determinado intervalo de tempo envia metade dos seus valores a um vizinho aleatório e a outra metade fica para si mesmo. Quando um nodo recebe uma mensagem de um vizinho, soma os pesos que recebeu na mensagem aos pesos que tinha guardado. Este algoritmo tem a tendência de convergir para o resultado correto, quanto maior for o número de mensagens trocadas por entre todos os nodos.

A soma dos pesos na rede têm de ser constante para que a haja convergência. No caso de uma mensagem ser perdida e não for recuperada, o resultado não vai convergir para o valor correto.

As principais vantagens deste algoritmo consistem no facto de ser fiável, convergir para o resultado correto, os cálculos efetuados em cada nodo serem muito reduzidos e funcionar para qualquer topologia. Por outro lado, o tempo para se obter o resultado depende do tamanho e topologia da rede. Se houver a falha de um nodo é muito provável que o resultado não convirja para o valor correto.

3.2 Implementação

Inicialmente foi implementado o algoritmo tal como tinha sido descrito em que foi assumido que não ocorreria perda de mensagens e que a topologia era constante. Para tal, era apenas necessário criar dois tipos de eventos, um que enviasse os pesos para um vizinho aleatório, o *gossip*, e um outro evento, o *iterator*, que em determinados períodos de tempo indicava que o nodo teria que enviar uma mensagem a um vizinho aleatório. Como o objetivo de cada *gossip* passava por enviar os pesos para um vizinho e não para toda a rede, então foi considerado que cada mensagem tinha um *target*, que não era alterado, mesmo que a mensagem fosse perdida ou a estrutura da rede fosse alterada.

Para tornar o algoritmo mais completo teríamos de o tornar imune à perda de uma parte das mensagens enviadas e à alteração da estrutura da rede em determinados intervalos de tempo. Esta alteração implica que sejam realizadas algumas verificações extra. Nomeadamente, sempre que um nodo envie uma mensagem do tipo *gossip* era originado o evento *collector*. Ao fim de um determinado *timeout* este evento enviava uma mensagem do tipo *ihave* aos vizinhos, caso o nodo que originou o *gossip* não tenha conhecimento que o *target* tenha recebido a mensagem. Desta forma, permitia poupar a quantidade de dados enviados pelo sistema, uma vez que o recetor já poderia ter os dados, mas o nodo que enviou o *gossip* originalmente pode ainda não ter conhecimento da receção da mensagem por parte do destino. O nodo que recebia um *ihave*, originava um evento *schedule*, que tinha um determinado *timeout*. Passado esse *timeout*, caso o nodo não soubesse que o *target* da mensagem já tinha recebido os valores, ia enviar um *request* a um vizinho que soubesse que tinha os dados, normalmente o vizinho que lhe tinha enviado o *ihave*. Para que os nodos tivessem conhecimento dos dados recebidos por cada nodo no sistema, criamos o tipo de mensagem *wehave* que contêm todos os *id* das mensagens que o próprio nodo conhece e quais os *id* que sabe que todos os nodos conhecem. Assim, poderia-se propagar pela rede o conhecimento de cada nodo e criar um *garbage collector* para eliminar dados desnecessárias. Estas mensagens eram geradas pelos eventos *knowledge* em determinados períodos de tempo.

4 Simulador

De forma a verificar a nossa implementação do algoritmo críamos um simulador que permitia correr um algoritmo. Para começar o simulador inicializava os pesos dos nodos conforme a função de agregação escolhida. Em seguida gerava os eventos iniciais, para posteriormente correr a simulação.

Na elaboração do simulador decidimos implementar algumas funcionalidades, para o tornar mais iterativo e mais fácil de experimentar diferentes tipos nomeadamente:

- a perda de mensagens entre nodos de forma aleatória conforme uma determinada probabilidade
- a reconstrução da rede em intervalos de tempo fixos
- a escolha da margem de erro para parar a simulação
- a contagem do número de mensagens perdidas
- a escolha do tempo que determinados eventos demoram a ser realizados
- vários tipos de eventos que eram requiridos pelo **Push-Sum Protocol**:
    1. *schedule*
    2. *collector*
    3. *knowledge*
    4. *iterator*

Uma outra funcionalidade que decidimos implementar foram os *snapshots*, que permitia ao simulador em determinados períodos de tempo guardar os valores de cada nodo de forma a verificar se o algoritmo estava com a tendência em convergir e consequentemente, se o algoritmo estava a ser eficaz.

5 Papel do simulador

O simulador que usaríamos para testar o algoritmo tinha o papel de ajudar a descobrir algumas falhas na nossa implementação, além da verificação de pequenas falhas na implementação, foi especialmente importante na fase de otimização.

As mensagens enviadas pelos eventos *wehave* eram dicionários em que a chave era o *id* de um nodo. Os valores associados a cada chave são os *ids* das mensagens que o nodo conhece. Se em cada evento do tipo *wehave* fossem enviados estes dicionários, ao fim de algum tempo era causado um grande *overhead*

na rede e muita da informação enviada era repetida. Para contornar este problema podíamos transformar a mensagem numa matriz. Nesta primeira implementação o valor era calculado a partir do valor que mais alto de todos os *ids* de um determinado nodo que tenha todos os número consecutivos a partir do zero.

Como uma mensagem *gossip* tem apenas um *target*, não havia a necessidade de enviar uma mensagem para todos os nodos e com a introdução de otimizações nas mensagens um nodo poderia nunca informar os outros nodos que já tinha uma determinada mensagem. Por exemplo, se um nodo recebesse a mensagem com *id* (0,1), poderia nunca chegar a informar os outros nodos que tinha este *id*, porque não era obrigatório receber a mensagem com *id* (0,0), visto que se o *target* da mensagem com *id* (0,0) já tivesse recebido a mensagem, esta não ia ser mais propagada pelo resto da rede. Ao descobrirmos esta falha modificamos ligeiramente como a otimização era feita. Nesta modificação as mensagens *wehave*, continuavam a ser matrizes, mas o seus valores eram calculados de forma diferente. Estes valores poderiam ser definido pela seguinte equação:

$$x = \sum 2^i \forall i \in Info \tag{1}$$

Em que *Info* era o conjunto de todos os *id* de um determinado nodo, associado a um emissor, por exemplo, se o nodo 0 tivesse os dados das mensagens (1,1) e (1,3) o *Info* seria [1,3] e o valor de x seria 10, que estaria na linha 0 coluna 1 da matriz.

Um exemplo mais completo do que realmente acontece é o caso seguinte em que o nodo 1 tem o seguinte conhecimento:

```
{
0: [(0,0), (0,1), (1,1)],
1: [(0,1), (1,0), (1,1), (1,2)]
2: [(0,0),(1,2),(2,0)]
}
```

com a compressão a informação enviada passaria para:

```
[
    [3, 2, 0],
    [2, 7, 0],
    [1, 4, 1]
]
```

6 Resultados

De forma a confirmar a viabilidade do algoritmo e do simulador decidimos testar e comparar os resultados de alguns dos valores dos diferentes testes.

Foram definidos vários valores que alteravam a dinâmica da simulação, mas entre estes decidimos verificar o impacto do valor do *loss*, este valor diz respeito à probabilidade de perder um pacote. Os testes foram realizados em redes em que o número de nodos variava entre 4 e 30, todas as ligações foram feitas aleatoriamente, o tempo de envio das mensagens era constante, para a função de agregação **COUNT**, em que eram geradas novas mensagens *gossip* de 15 em 15 unidades de tempo e a simulação era terminada assim que todos os nodos tivessem um erro inferior a 1%.

Para cada um dos números de nodos foram realizados dez testes para se obter resultados mais precisos e guardados os valores das mensagens perdidas, o número de mensagens trocadas, e o número de rondas que demorou para convergir, para mais tarde fazer uma média de cada um destas medidas. Para o último teste da simulação foi guardado o valor dos *snapshots*.

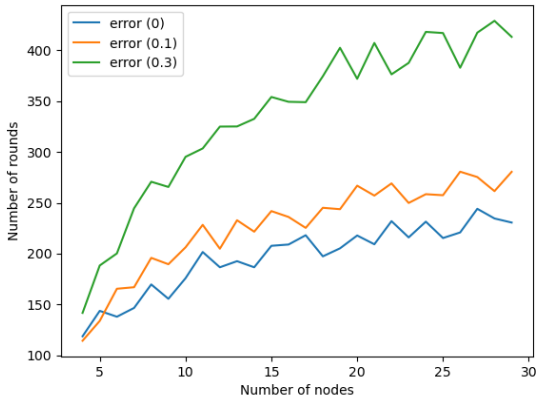

Figura 1: Tempo médio para convergir

A figura 1 mostra o tempo médio para convergir para cada uma das redes com diferente número de nodos. Como era esperado um maior número de mensagens perdidas representa um aumento significativo no tempo para convergir. Para redes com um número de nodos mais alto, uma perda de 30% das mensagens duplicou o tempo necessário para convergir.

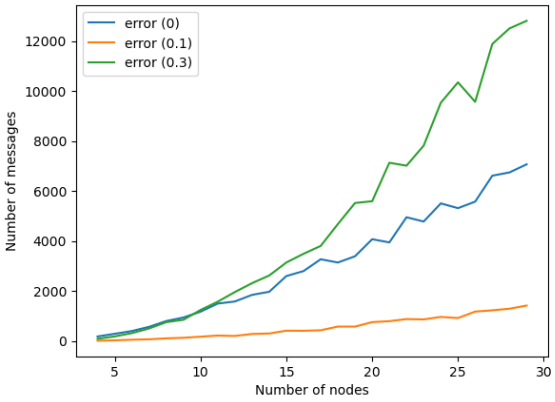

Figura 2: Número médio de mensagens enviadas

Em relação ao número de mensagens trocadas entre nodos, a figura 2 mostra a variação para diferentes probabilidades de perda de mensagens. Uma perda de 10% e 30% das mensagens implicou um aumento de 3 e 10 vezes do número de mensagens enviadas, respetivamente.

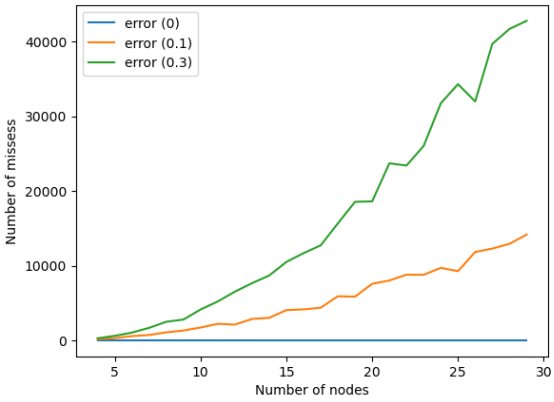

Figura 3: Número médio de mensagens perdidas

Ao compararmos a figura 2 com a figura 3, verificamos que a diferença entre o número de mensagens trocadas e o número de mensagens perdidas é significativamente superior ao número de mensagens perdidas, pelo que podemos concluir que a perda de uma mensagem pode significar o envio de várias mensagens para compensar essa perda.

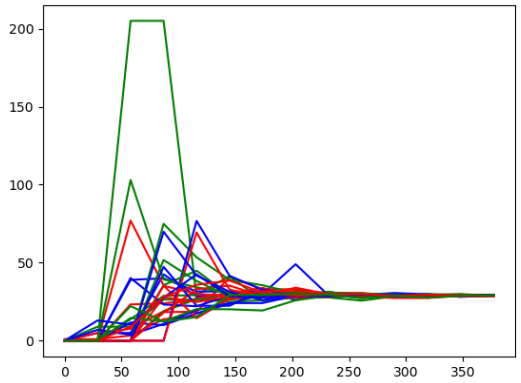

Figura 4: Valor dos pesos de todos os nodos ao longo do tempo

A figura 4 mostra a evolução dos pesos de todos os nodos de uma rede com 30 nodos. A função de agregação utilizada foi a **COUNT** e como podemos ver o peso de todos os nodos encontra-se aproximadamente no 30 no final da simulação. Para que todos os nodos tivessem um erro inferior a 1% foi necessário esperar praticamente 400 unidades de tempo, mas por volta das 250 o erro já era bastante reduzido. Nesta simulação a probabilidade de perda de pacotes era 30% e a rede era alterada de 15 em 15 unidades de tempo.

7 Conclusão

Neste relatório descrevemos o algoritmo que utilizamos para fazer as funções de agregação, o porquê do o termos escolhido, algumas implementações do simulador e de que forma este nos ajudou a verificar a implementação do algoritmo. Por fim mostramos que o simulador implementado em conjunto com o

algoritmo são capazes de calcular com alguma aproximação o valor desejado, como inicialmente tínhamos previsto.

De destacar que os teste feitos poderiam ter sido um pouco mais abrangentes, nomeadamente podiamos ter utilizado topologias diferentes e redes com um número de nodos muito superior.

Referências

[1] Paulo Jesus, Carlos Baquero, and Paulo Sérgio Almeida. A Survey of Distributed Data Aggregation Algorithms. *arXiv:1110.0725 [cs]*, October 2011. arXiv: 1110.0725.