JHEFERSON RENAN BORGES

VISUAL VCUBE: UMA IMPLEMENTAÇÃO GRÁFICA DO ALGORITMO DE DIAGNÓSTICO DISTRIBUÍDO VCUBE

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Elias P. Duarte Jr.

CURITIBA PR

2019

RESUMO

Sistemas distribuídos são constituídos de um conjunto de processos em execução que se comunicam utilizando troca de mensagens e executam em computadores em uma rede. Como são cada vez mais importantes, é necessário garantir que os sistemas distribuídos sejam tolerantes a falhas. O primeiro passo para um sistema ser tolerante a falhas é a capacidade de detectar quando ocorre uma falha. Para realizar essa tarefa podemos usar o diagnóstico distribuído. Com o diagnóstico distribuído cada processo pode descobrir o estado de cada elemento que compõe o sistema distribuído, realizando testes entre si. Existem diversos algoritmos de diagnóstico, entre eles destaca-se o VCube, um algoritmo escalável por possuir diversas propriedades logarítmicas. O intuito desse trabalho é facilitar a compreensão das funcionalidades e vantagens do VCube com uma implementação gráfica, chamada Visual VCube. Para efeito de comparação, além do VCube, foram também implementados algoritmos de diagnóstico baseados em broadcast, gossip e anel. A implementação foi feita em JavaScript usando a biblioteca gráfica P5, que pode ser reproduzida em qualquer navegador Web.

Palavras-chave: Diagnóstico. Algoritmos Distribuídos. Visualização de Algoritmos Distribuídos

SUMÁRIO

1 INTRODUÇÃO

Sistemas distribuídos fazem parte da nossa realidade sendo usados em diversas sistemas usados no nosso dia-a-dia. Além de diversas aplicações em redes de computadores e contextos específicos como bancos de dados distribuídos, algoritmos de roteamento e aplicações *peer-to-peer*. Os sistemas distribuídos são usados também em outros contextos como controle de processos em tempo real, controle trafego aéreo e controle industrial [CW11]. Alguns sistemas de computação paralela também são sistemas distribuídos, em particular os baseados em MPI [Pac96], sendo muito importantes na computação científica.

Sistemas distribuídos são constituídos de um conjunto de processos em execução que se comunicam utilizando troca de mensagens e executam em computadores em uma rede [GR10]. Neste trabalho processos são também chamados de nodos. Mas, para poder utilizá-los, de forma ideal, é necessário garantir que eles sejam tolerantes a falhas. O primeiro passo para um sistema ser tolerante a falhas é a capacidade de detectar quando ocorre uma falha. Para realizar essa tarefa podemos usar o diagnóstico distribuído. Com o diagnóstico distribuído cada processo pode descobrir o estado de cada elemento que compõe o sistema distribuído, realizando testes entre si.

Existem diversos algoritmos para implementar o diagnóstico distribuído. O mais simples é o algoritmo força bruta baseado em broadcast (difusão) no qual todos os processos testam todos os demais processos gerando uma grande quantidade de mensagens, porém realizando o diagnóstico no menor tempo possível.

Outra estratégia para diagnóstico é o algoritmo em anel onde cada processo testa seus sucessores até encontrar um processo sem falha [BB92]. Este algoritmo utiliza a menor quantidade de mensagens possível, mas aumenta o tempo para completar diagnóstico.

Além destes, no algoritmo baseado em disseminação probabilística (gossip) [GR10] cada processo testa uma quantidade fixa de nodos escolhidos aleatoriamente, o que resulta em uma baixa quantidade de mensagens, mas só garante probabilisticamente quando termina o diagnóstico .

Por fim, o algoritmo VCube [DBR14] surge como uma solução que busca uma menor quantidade de testes entre os processos do sistema e mesmo assim mantém um tempo reduzido para que qualquer evento correspondendo a mudanças de estado dos processos seja descoberta por todos os processos sem falhas. Para isso utiliza uma topologia semelhante ao hipercubo para realização dos testes. Mesmo com nodos falhos, o VCube apresenta diversas propriedades logarítmicas.

O intuito desse trabalho é facilitar a compreensão das funcionalidades e vantagens do VCube com uma implementação gráfica, chamada Visual VCube. Essa implementação foi feita em JavaScript usando a biblioteca gráfica P5 [p5], que pode ser reproduzida em qualquer navegador Web.

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta os algoritmos de diagnóstico implementados. O Capítulo 3 detalha a implementação do Visual VCube e, por fim, o Capítulo 4 conclui o trabalho.

2 DIAGNÓSTICO DISTRIBUÍDO

Nas próximas seções são apresentados o modelo de sistema e as estratégias de diagnóstico implementadas no trabalho.

2.1 MODELO DE SISTEMA

Considere um sistema distribuído de N processos totalmente conectados, ou seja, existe um canal de comunicação entre cada par de processos. Considere também que um processo pode estar falho ou sem-falha. Os processos executam um algoritmo de diagnóstico para determinar o estado do sistema, isto é, o estado de todos os processos. Na realização do diagnóstico, os processos executam testes entre si. Um teste corresponde à aplicação de um estímulo e obtenção da resposta correspondente. Se um processo é testado como sem-falha, o testador pode obter informações de diagnóstico. As falhas são por parada (*crash*) portanto um processo falho não responde a nenhum estímulo.

Uma rodada de testes é definida como o tempo necessário para que todos os processos realizem seus testes, assinalados pelo algoritmo de diagnóstico e obtenham as respostas destes. A latência é definida como a quantidade de rodadas de testes que são necessárias para que todos os processos sem-falhas descubram a alteração de estado de um processo. Outra métrica importante é a quantidade de mensagens que um circulam no sistema. Considerando que um processo envia uma mensagem para cada teste, essa quantidade está relacionada a quantos processos cada processo testa. Essas métricas serão usadas para demonstrar as vantagens e desvantagens de cada algoritmo de diagnóstico.

2.2 ALGORITMO BASEADO BROADCAST

No algoritmo baseado em broadcast todo processo sem-falha testa todos os processos do sistema. Para cada teste, o testador ao receber a resposta sabe que o processo testado não está falho. Agora, se o testador não receber a resposta em um tempo estipulado, conclui que o processo testado está falho. Assim, para um processo i descobrir o estado de um processo j basta realizar um teste. Como todos os processos testam todos os demais processos em todas as rodadas de teste, basta uma rodada para completar o diagnóstico. Este algoritmo portanto possui a menor latência possível mas, em compensação, necessita de N^2 mensagens por rodada.

2.3 ALGORITMO EM ANEL

Partindo do objetivo de minimizar a quantidade mensagens necessarias para completar o diagnóstico foi proposto o algoritmo em anel *Adaptive Distributed System-level Diagnosis (Adaptive-DSD)* [BB92]. Neste algoritmo cada processo possui um antecessor e um sucessor. Cada processo testa o sucessor, se este estiver falho, testa o sucessor do sucessor, e assim sucessivamente até encontrar um processo sem-falha, ou testar todos os processos falhos. Para um processo i descobrir o estado do processo j ele ou testa diretamente j ou obtém a informação de um processo testado sem-falha. O algoritmo garante que só existe um caminho pelo qual a informação chega a um processo que consiste de um anel conectando todos os processos sem-falhas. Esse algoritmo necessita de exatamente N mensagens para o realizar o diagnóstico,

uma vez que cada processo é testado uma vez por rodada pelo primeiro antecessor que sem-falha. Em compensação, a latência é de N rodadas no pior caso [Ruo13].

2.4 ALGORITMO BASEADO EM GOSSIP

No algoritmo baseado em Gossip, todos os processos selecionam k processos aleatoriamente para testar, dentre todos os processos do sistemas. O parâmetro k é chamado de *fanout*. Um *fanout* maior diminui a quantidade de rodadas necessárias para completar o diagnostico. Entretanto, com o incremento do *fanout* se aumenta a quantidade de mensagens [GR10]. No algoritmo baseado em Gossip um processo i obtém informações sobre um processo j de forma probabilística. A quantidade de mensagens utilizadas por rodada é k multiplicado por N. Esse algoritmo não garante a latência máxima, mas a partir de uma certa quantidade de rodadas, que depende de k e N, existe uma alta probabilidade de todos os processos completarem o diagnóstico de um evento.

2.5 ALGORITMO VCUBE

No algoritmo VCube [DBR14] o diagnóstico ocorre de forma hierárquica formando um hipercubo quando todos os processos estão sem-falhas. Para a realização dos testes, os processos são separados em *clusters*. Um *cluster* é uma partição de processos com tamanho s variando de 1 a $\log_2 N$. O processos a serem testados pelo processo i no *cluster* de tamanho s são definidos pela função $C_{i,s}$: $C_{i,s} = i \oplus 2^{s-1} || C_{i \oplus 2^{s-1},k} | k = 1, 2, ..., , s-1$.

Onde s é o indice do *cluster* onde o teste será realizado e possui valor entre 1 e $\log_2 N$. No algoritmo, cada um dos N processos testa seus s *clusters*. Para garantir que cada processo é testado por um único processo de cada *cluster*, o processo j só é testado pelo primeiro processo sem-falha de $C_{j,s}$, evitando assim o testes excessivos em condições especificas.

Ao testar um processo sem-falha, o testador obtém informações de diagnóstico a partir deste. Para garantir que as informações sejam atualizadas corretamente é usado um *timestamp* associado a cada estado. Se o *timestamp* do testador for menor que o da informação obtida o testador altera o estado local.

A quantidade de mensagens utilizadas peloo VCube é no máximo $Nlog_2N$ e a latência no pior caso é de log_2^2N rodadas de testes. Como são funções logarítmicas o VCube pode ser considerado um algoritmo escalável.

3 IMPLEMENTAÇÃO GRÁFICA

Este capítulo descreve como foi realizada a implementação da representação gráfica dos algoritmos de diagnóstico incluindo as bibliotecas utilizadas, as funcionalidades e os detalhes de implementação.

3.1 DESCRIÇÃO DA REPRESENTAÇÃO GRÁFICA DE DIAGNÓSTICO

Neste trabalho os algoritmos foram implementados de forma gráfica. Na representação gráfica um nodo é representado por um círculo numerado. Um nodo sem-falhas se apresenta a cor azul e com falhas é representado na cor vermelha. Em cada nodo são indicados os demais nodos que este nodo considera falho apresentando seus identificadores logo abaixo do identificador do nodo em questão. As mensagens são representadas por círculos verdes e as respostas são representadas como quadrados amarelos, como ilustrado na figura 3.1. As mensagens se movem do nodo testador ao nodo testado.

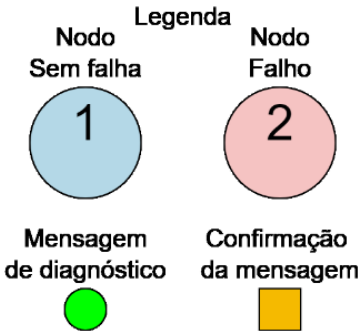

Figura 3.1: Legenda da representação.

Além disso, na implementação do VCube temos retângulos que englobam vários nodos que representam os *clusters*, como ilustrado na figura 3.2.

Figura 3.2: Cluster de tamanho 2.

Na representação topológica de todos os algoritmos, exceto o VCube, os nodos são distribuídos em uma elipse de forma que as mensagens não colidam durante o trajeto, como na mostrado na figura 3.3. Na representação do VCube, nodos de um *cluster* estão sempre próximos uns dos outros para ficar clara a organização destes, como mostrado nas figuras 3.4 e 3.5. Para evitar colisões nesse formato algumas mensagens realizam uma trajetória não linear.

3.2 A BIBLIOTECA P5.JS

A biblioteca P5.js [p5] é uma biblioteca de código aberto JavaScript usada para criar desenhos e animações gerando HTML (*Hyper Text Markup Language*). A biblioteca cria um

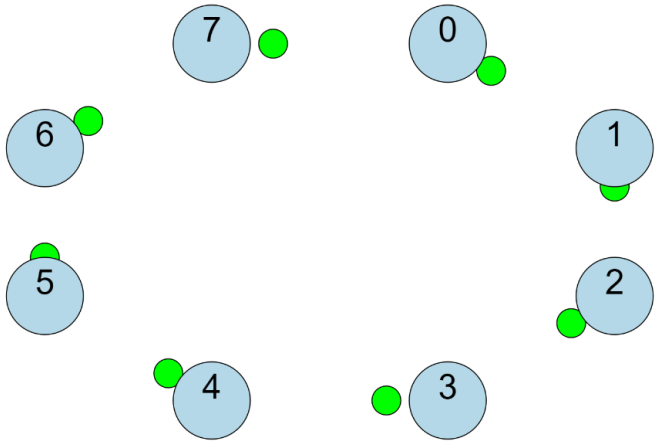

Figura 3.3: Representação topológica elíptica.

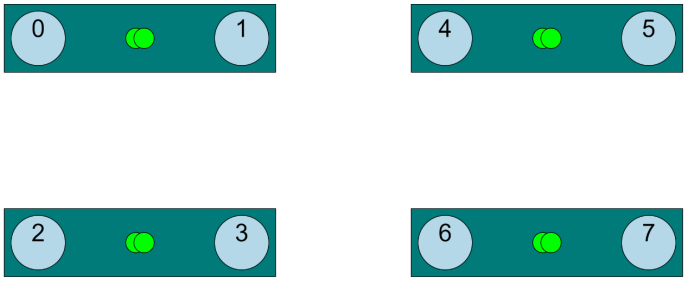

Figura 3.4: VCube 8 nodos em *clusters* de tamanho 2.

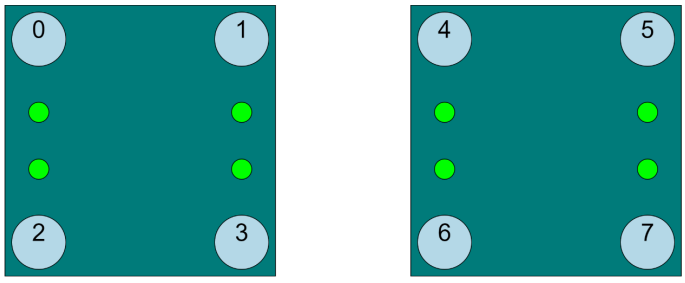

Figura 3.5: VCube 8 nodos em *clusters* de tamanho 4.

quadro branco (*canvas*) e neste é possível criar e mover objetos como retângulos e círculos. Além disso, fornece a interface necessária para tratar cliques do mouse possibilitando assim a interação com o usuário. A biblioteca funciona em duas fases, descritas a seguir.

A primeira fase consiste da função *setup()* que é executada uma vez na inicialização da página HTML gerada. Na função é definido o tamanho da visualização, uma quantidade inicial de elementos e o algoritmo que vai executar.

A segunda fase funciona com a função *draw()* que é executada periodicamente, sendo esta responsável pelas atualizações da visualização a cada intervalo de tempo. No contexto deste trabalho, utilizando esta função são representados os nodos, as mensagens e os textos que são exibidos.

3.3 A IMPLEMENTAÇÃO

A implementação foi realizada em Javascript, e está disponível em **github.com/jhefborges/virtual-vcube**.

A implementação se divide em um programa principal (**sketch.js**) que utiliza a biblioteca P5.js e as classes criadas para representar os algoritmos de diagnóstico. O programa **sketch.js** importa a biblioteca P5.js e utiliza as funções setup() e draw() para representar graficamente os algoritmos. As classes são de dois tipos: um representa os nodos e o outro representa os algoritmos. A classe que representa os nodos é a **Nodos.js** que contém a coleção de todos os nodos do sistema. As classes que representam os algoritmos estendem a classe **Base.js** e implementam os métodos que especificam os testes.

A documentação foi gerada a partir JSDoc, que gera páginas HTML com as informações do código a partir de comentários.

3.4 FUNCIONALIDADES

As funcionalidades do Visual VCube permite três tipos de representação visual. A legenda, ilustrada na figura 3.1 mostra os componentes gráficos utilizados. A representação guiada é aquela que demonstra a execução dos algoritmos e contém observações em texto que destacam o comportamento de cada algoritmo. Possui sempre 8 nodos, após a primeira rodada de testes o nodo 6 falha para demonstrar a disseminação de informações sobre o evento em cada algoritmo. Por fim o tipo de representação interativa é definido somente para o algoritmo VCube. Esta representação permite que o usuário falhe nodos durante a execução. O sistema pode possuir entre 2 e 32 nodos e inicia com todos os nodos sem-falhas. Ao clicar em um nodo sem-falha este se torna falho e, analogamente se o nodo estiver falho ele se recupera.

4 CONCLUSÃO

O diagnóstico distribuído permite que os processos de um sistema distribuído detectem falhas no sistema. A detecção de falhas é uma etapa essencial para a construção de sistemas tolerantes a falhas. Neste trabalho foi apresentada uma implementação gráfica de diversos algoritmos de diagnóstico: baseados em broadcast, gossip e anel, além do VCube. O VCube é um algoritmo escalável por possuir diversas propriedades logarítmicas. O objetivo do trabalho é facilitar a compreensão das funcionalidades e vantagens do VCube. A implementação foi feita em JavaScript usando a biblioteca gráfica P5, que pode ser reproduzida em qualquer navegador Web.

Trabalhos futuros incluem implementar a representação interativa para os demais algoritmos, além do VCube. Na verdade a funcionalidade está presente, mas precisa apenas ser mais testada. Além disso podem ser implementados de forma gráfica os diversos algoritmos que foram definidos sobre o VCube.

REFERÊNCIAS

[BB92] R. Bianchini and R. Buskens. Implementation of online distributed system-level diagnosis theory. *IEEE Transactions on Computers*, 41(05):616–626, 1992.

[CW11] J. Coulouris and P. Wesley. *Distributed Systems: Concepts and Design*. Pearson, 2011.

[DBR14] E. P. Duarte, L. C. E. Bona, and V. K. Ruoso. Vcube: A provably scalable distributed diagnosis algorithm. In *The 5th IEEE Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, pages 17–22, 2014.

[GR10] R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer, 2010.

[p5] P5.js: Javascript library for creative coding. `https://p5js.org/`. Acessado em 27/11/2019.

[Pac96] P. S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[Ruo13] V. K. Ruoso. Uma estratégia de testes logarítimica para o algoritmo hi-adsd, Dissertação de Mestrado, Pós-Graduação em Informática - Universidade Federal do Paraná, 2013.