Trabalho Prático 3

Atualizações Problemáticas

1. Introdução

O trabalho prático tem por objetivo solucionar o problema de uma empresa que mantém um aplicativo que sempre precisa estar disponível para seus usuários. Seus servidores conectados entre si precisam ser ocasionalmente atualizados. sendo que quando um servidor fica indisponível seu tráfego é redirecionado para seus vizinhos. As informações fornecidas para a solução do problema são a quantidade de servidores e as conexões existentes. Através destas informações é preciso encontrar a quantidade mínima de rodadas de atualização, bem como a determinação dos servidores selecionados para cada rodada.

O trabalho exige a implementação de duas soluções diferentes: uma heurística, que determina uma solução não garantidamente mínima mas válida, e um algoritmo exato, que produz a menor quantidade de rodadas dado o cenário da rede. Ambas as soluções utilizam o conceito de coloração de grafos, que consiste na determinação de cores para os vértices de um grafo de tal forma que dois vértices adjacentes não possuam a mesma cor.

2. Modelagem do problema

A rede de servidores pode ser modelada em um grafo não orientado e não ponderado, em que os vértices são os servidores e as arestas representam uma conexão entre dois servidores. A estrutura utilizada para armazenar o grafo foi de lista de adjacências. A construção desta estrutura pode ser feita à medida que as entradas vão sendo coletadas.

A determinação da solução está diretamente relacionada com o problema de coloração de grafos. As cores determinadas para cada vértices correspondem à rodada que o servidor será atualizado. A menor quantidade de rodadas corresponde à menor quantidade de cores que pode ser utilizada para colorir o grafo. A alocação para as rodadas correspondem aos vértices que estão com a mesma cor após o término da execução do algoritmo.

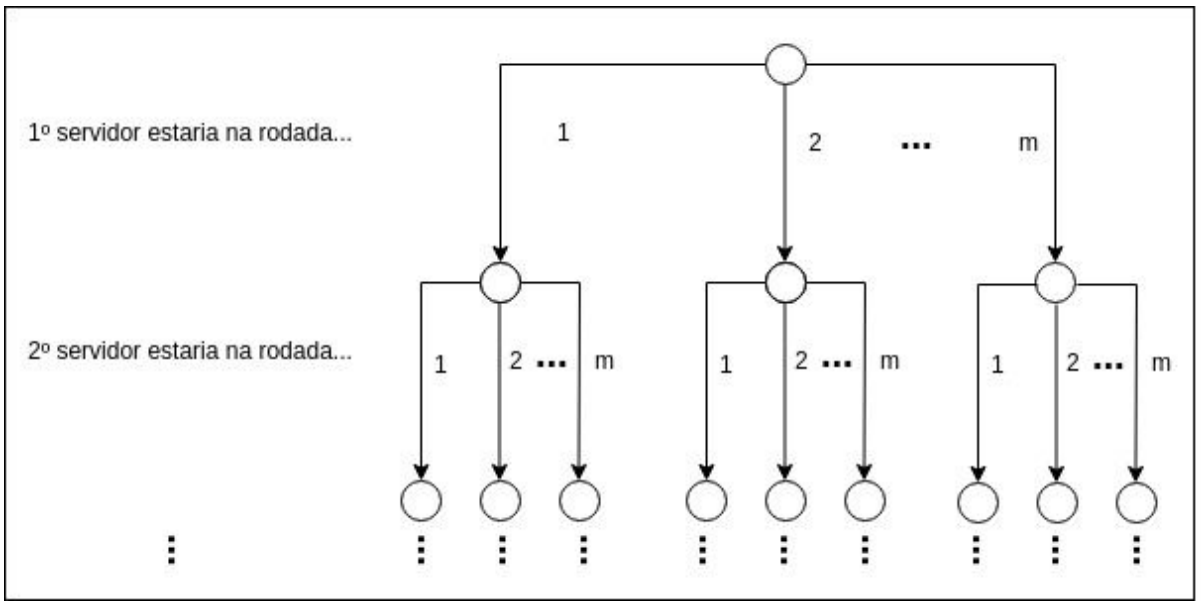
2.1. Força bruta

Pela especificação do problema, a empresa inicialmente possui poucos servidores. Isso configura um cenário em que é aceitável a determinação exata do número mínimo de rodadas através de um algoritmo de força bruta. O problema de otimização “Qual a menor quantidade de rodadas para se fazer a atualização?” pode ser reduzido ao problema de decisão “É possível realizar a atualização em m rodadas?”.

2.1.1. É possível realizar a atualização em m rodadas?

Para resolver o problema de decisão precisamos calcular todas as possibilidades de alocação dos servidores para as m rodadas, resultando em m^V possibilidades de alocação, sendo que V é o número de servidores (ou número de vértices).

Definida uma ordem para os servidores (por exemplo, id do servidor). Podemos gerar tais soluções possíveis formando uma árvore em que no primeiro nível se encontra o primeiro servidor. À cada nível da árvore estamos variando a rodada de 1 até m em que o servidor estaria alocado e assim por diante. A imagem a seguir ilustra a árvore de soluções formada:



Árvore de solução que gera todas as possibilidades de alocação dos servidores nas m rodadas

No último nível da árvore teremos possíveis soluções de alocação dos V servidores. Para checar se uma solução é válida basta verificar se não há dois vértices adjacentes alocados para a mesma rodada. Uma vez encontrada uma solução válida o problema está resolvido.

Uma poda pode ser feita para melhorar em alguns casos a determinação da solução exata. À cada nível uma checagem rápida pode ser feita para ver se o vértice atual já possui vizinhos alocados para a mesma rodada. Caso tenha, não é preciso continuar construindo a solução uma vez que já é invalidada por este vértice.

2.1.2. Qual a menor quantidade de rodadas para se fazer a atualização?

Podemos solucionar o problema de otimização utilizando o problema de decisão anterior, basta verificar qual o menor número de rodadas necessário para se obter uma solução válida. Começando com 1 rodada, se testa o problema “É possível realizar a atualização em 1 rodada?”, e assim por diante até que se encontre uma solução válida.

2.2. Heurística

Com o aumento do número de servidores da empresa a solução por força bruta se torna inviável. Utilizando uma heurística podemos encontrar uma alocação válida porém não garantidamente com uma quantidade mínima de rodadas possível para realizar a atualização.

A solução heurística dada é um algoritmo guloso. Novamente defina uma ordem para os servidores. Começando do primeiro servidor, itera-se sobre os servidores definindo a menor rodada que o servidor pode ser alocado. A determinação da menor rodada consiste na verificação das rodadas em que seus vizinhos estão alocados. Ao final da iteração teremos uma solução vaĺida para o problema. A quantidade mínima de rodadas corresponde à maior rodada na qual um vértice foi alocado.

3. Análise Teórica do Custo Assintótico

3.1. Análise Teórica do Custo Assintótico de Tempo

Para se construir o grafo a partir das informações da rede de servidores é preciso criar um vetor de tamanho V para se armazenar os vértices, além de inicializá-los com algum valor inicial para a rodada. Logo após, é preciso inserir as E arestas nas listas de adjacência. As operações de criação do vetor, inicialização de um vértice, criação das listas de adjacência e a inserção de uma nova aresta têm complexidade constante $O(1)$. Assim, a etapa inicial de construção tem complexidade de tempo $O(V + E)$.

O registro do resultados e da alocação obtida são armazenados em 2 arquivos de saída. Para cada vértice é preciso imprimir uma linha com a rodada para o qual foi alocado. Considerando que a escrita em um arquivo tem custo constante $O(1)$, esta etapa tem complexidade de tempo $O(V)$.

Para liberar o espaço alocado para a estrutura da rede de servidores é preciso liberar a memória de cada vértice e de suas respectivas arestas. Assim, a complexidade de tempo deste passo é $O(V + E)$.

3.1.1. Força bruta

Como foi dito na modelagem do problema, a solução por força bruta gera m^V possíveis soluções para o problema. No pior caso, a poda não pôde ser aplicada e todas as possíveis soluções precisam ser avaliadas.

Para avaliar a validade de uma solução, é preciso iterar sobre vetor de vértices de tamanho V. Para cada vértice é preciso verificar se algum de seus vizinhos está na mesma rodada. No final da verificação, o algoritmo terá iterado sobre V vértices e percorrido $2 \times E$ arestas. A complexidade para se verificar uma solução é $O(V + E)$.

Assim, a complexidade do algoritmo de força bruta é $O(m^V \times (V + E))$. Este é o maior gargalo do programa, superando assintoticamente todas as outras etapas anteriores e posteriores.

3.1.2. Heurística

O algoritmo guloso itera sobre um vetor de vértices de tamanho V. Para cada vértice é preciso achar a menor rodada na qual este vértice pode ser alocado.

A estratégia adotada para encontrar a menor rodada na qual um vértice pode ser alocado é a seguinte. Comece tentando alocar o vértice na rodada 1. Verifique se algum vizinho já está alocado para esta rodada. Caso esteja, incremente a rodada. Faça até que nenhum vizinho esteja alocado para a rodada atual. No pior caso, precisaríamos de colocar o vértice em uma rodada maior do que todas as de seus vizinhos. Considerando que o grau D de um vértice é a quantidade de vizinhos que este vértice possui, a complexidade para este passo é $O(m \times D)$.

No pior caso da heurística, ao final da iteração sobre o vetor de vértices teremos verificado $m \times E$ vezes se um vértice está na rodada certa ou não. Assim, a complexidade final do algoritmo guloso é $O(V + m \times E)$. Este é o maior gargalo do programa, superando assintoticamente todas as outras etapas anteriores e posteriores.

3.2. Análise Teórica do Custo Assintótico de Espaço

A única estrutura utilizada em ambas as soluções (força bruta e heurística) é o grafo. A implementação utiliza a forma de lista de adjacências, tendo um vetor com V posições para cada servidor. Em cada posição é guardada uma lista encadeada com as arestas que saem do vértice, somando um total de E arestas representando as conexões entre os servidores. Assim, a complexidade assintótica de espaço do programa se dá pelo custo de armazenamento desta estrutura, que é $O(V + E)$.

4. Análise do Experimento

O experimento feito utiliza os casos de teste fornecidos pelo monitor. Os dados de memória utilizada e tempo gasto para a execução foram coletados da saída do script “run_tests.sh”. O experimento foi feito usando uma máquina com processador Intel CoreTM i7-3630QM CPU 2.40GHz × 8 e memória RAM de 7,5 GiB.

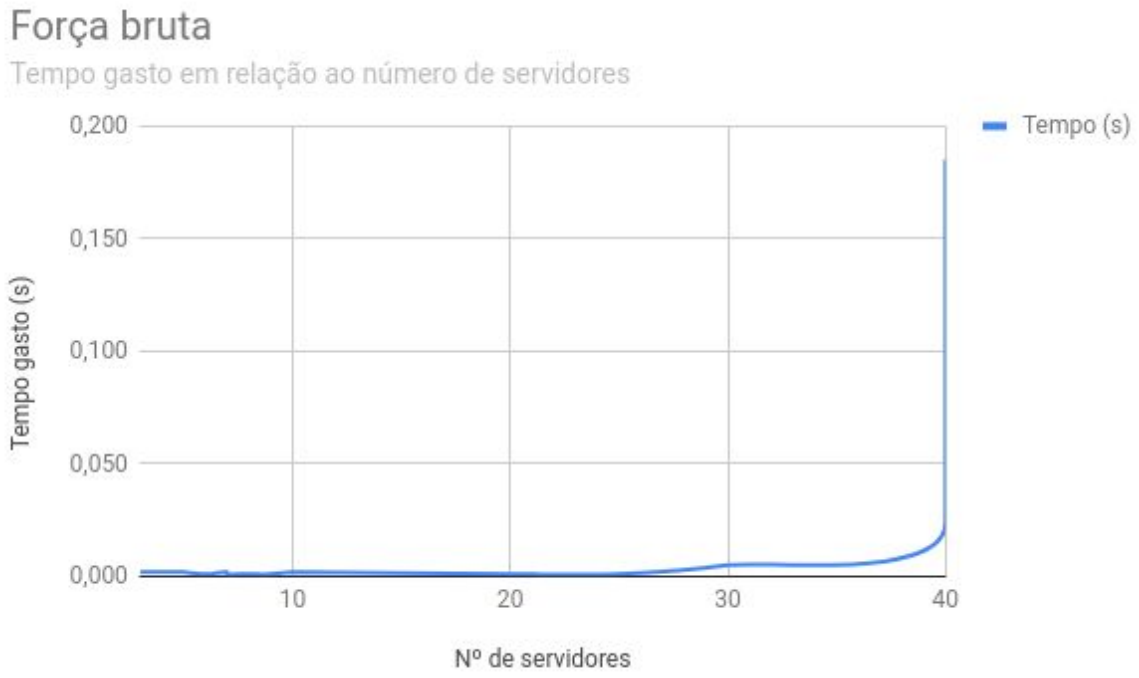
As execuções foram feitas de tal forma que conseguiremos ver variações de m (número de rodadas), V (número de servidores) e E (número de conexões). Tais variações influenciam o tempo de execução do programa, diminuindo ou aumentando de acordo com seus valores. Para evitar valores que não representem a fielmente o tempo gasto foram feitas 5 execuções para cada caso de teste, sendo tirada a média dos 5 tempos para se chegar ao tempo gasto em cada ponto.

4.1. Força bruta

Os casos de teste que a solução conseguiu computar o resultado exato em tempo hábil foram 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 17, 19, 22 e 26. Somente estes casos estão sendo utilizados para analisar o tempo de execução.

Relembrando, a complexidade final do programa que utiliza o algoritmo de força bruta foi da ordem de $O(m^V \times (V + E))$. Assim, o tempo de execução pode variar de acordo com o número de rodadas necessárias, o número de servidores e o número de conexões na rede.

Pelo fator m^V, podemos esperar um comportamento exponencial do tempo gasto em relação ao número de servidores e ao número de rodadas. Tal comportamento esperado pode ser observado nos dois gráficos abaixo, em que o tempo de execução pode ser visto em relação ao número de servidores e de rodadas separadamente.



Pelo fator multiplicativo $V + E$, podemos esperar um comportamento linear ao número de conexões, uma vez que o número de servidores já causa o aumento exponencial do tempo gasto. Tal comportamento esperado pode ser observado no gráfico abaixo, em que o tempo de execução parece variar linearmente com o aumento do número de conexões entre os servidores.

A partir deste experimento podemos confirmar a análise de complexidade de tempo teórica feita anteriormente. Podemos concluir também que a utilização desta solução se torna

inviável quando as entradas ganham escala, tendo que recorrer à solução heurística para determinar as rodadas de atualização em cenários com um número grande de servidores.

4.2. Heurística

A solução heurística conseguiu computar os resultados aproximados para todos os casos de teste. Assim, quase todos os casos de teste puderam ser utilizados no experimento. O caso de teste 25 teve um tempo gasto fora do esperado, causando uma variação anormal nos resultados. Por este motivo, ele não foi utilizado para a análise experimental.

Relembrando, a complexidade final do programa que utiliza a heurística foi da ordem de $O(V + m \times E)$. Assim, o tempo de execução pode variar de acordo com o número de rodadas necessárias, o número de servidores e o número de conexões na rede.

Pela parcela V, podemos esperar um comportamento assintótico linear ao aumento do número de servidores. Tal comportamento pôde ser observado ao realizar os experimentos variando o número de servidores. O gráfico abaixo mostra os resultados das execuções para os diferentes casos de teste:

Pela parcela $m \times E$, podemos esperar um comportamento “quadrático” do tempo de execução ao variar tanto o número final de rodadas quanto o número de conexões entre os servidores. Tal comportamento pôde ser observado ao realizar os experimentos variando estes valores. O gráfico abaixo mostra os resultados das execuções para os diferentes casos de teste:

A partir deste experimento podemos confirmar a análise de complexidade de tempo teórica feita anteriormente. Podemos concluir também que a utilização desta solução se torna viável quando as entradas ganham escala, solucionando o problema de forma aproximada porém em tempo hábil para a empresa determinar suas rodadas de atualização.

5. Comparação Heurística vs. Força bruta

Para comparar a eficiência da solução heurística foram gerados casos de teste que poderiam ser resolvidos pela solução exata. Foram gerados 100 casos de teste com número de servidores aleatório entre 2 e 10 vértices e quantidade de conexões aleatória que pode assumir valores entre 1 e 900, dependendo da quantidade de servidores.

À cada conjunto de testes gerados, foram executadas as duas soluções de força bruta e heurística. No total, foram gerados 10 conjuntos de testes diferentes. Em média, a solução heurística errou 9 casos de teste do conjunto gerado. Isso nos dá uma taxa de acerto de 91% da solução heurística nos experimentos realizados. Sobre os resultados errados da solução heurística, a diferença da quantidade de rodadas para a solução exata foi de no máximo 3 rodadas.

6. Conclusão

A heurística gulosa foi uma solução pertinente para este problema. A alta complexidade do cálculo no algoritmo de força bruta faz com que o tempo de execução cresça exponencialmente, se tornando impraticável para a empresa BH Software. Mesmo os resultados não sendo garantidamente mínimos, a aproximação mostra ser efetiva na comparação com o algoritmo exato. Com esta solução, os servidores poderão ser atualizados de forma a deixar o aplicativo funcionando sem interrupções, cumprindo assim o objetivo do trabalho.