Uso de Particionamento de Dados para Acelerar Simulações de Cidades Inteligentes

Francisco Wallison Rocha[1], **Emilio Francesquini**[2], **Daniel Cordeiro**[1]

[1] Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

{wallison.rocha, daniel.cordeiro}@usp.br

[2]Centro de Matemática, Computação e Cognição – Universidade Federal do ABC (UFABC)
Santo André – SP – Brasil

e.francesquini@ufabc.edu.br

Abstract. *Simulation is an invaluable technique to smart cities research. However, city-scale simulators can be processing and memory intensive and hard to scale. This work presents a load balancing algorithm for a distributed traffic simulator for smart cities.*

Resumo. *Simulação é uma técnica de grande valia para pesquisas na área de cidades inteligentes. Entretanto, simuladores capazes de simular eventos em cidades inteiras podem exigir muita memória e poder de processamento. Este trabalho apresenta um algoritmo de balanceamento de carga para um simulador de tráfegos concebido para o estudo de cidades inteligentes.*

1. Introdução

Megalópoles como São Paulo, Rio de Janeiro, Nova York e Tóquio sofrem com problemas decorrentes do acelerado crescimento populacional. Problemas relacionados a transporte público, distribuição de água, poluição, saúde, entre outros, requerem soluções inovadoras e não triviais de serem avaliadas antes de serem postas em prática [Santana et al. 2017]. Pesquisas em Cidades Inteligentes propõem o uso de simuladores para avaliar tais soluções. Estes podem criar cenários realistas de grandes megalópoles, porém, que podem ser difíceis de serem simulados devido ao poder computacional necessário.

O *InterSCSimulator* é um simulador escrito na linguagem baseada em atores Erlang. O uso da linguagem Erlang facilita a computação paralela e distribuída, mas não é suficiente para permitir a simulação de grandes cenários representativos de megalópoles. Tais cenários acarretam o uso excessivo de memória devido ao tamanho e complexidade dos cenários, um *speedup* não escalável e uma taxa de utilização da CPU de cerca de 50%, o que não é considerado ótimo [Santana et al. 2017].

O simulador foi implementado para funcionar em arquiteturas de memória compartilhada ou distribuída. No entanto, sua implementação distribuída baseia-se em uma estrutura de dados fornecida pelo ambiente Erlang/OTP chamada *ETS Table*, que não é distribuída. Para melhorar a escalabilidade do simulador é necessário melhorar a divisão de tarefas (troca de mensagens e viagens realizadas pelos atores) nas diferentes máquinas utilizadas pelo simulador *InterSCSimulator*.

2. *Binary Graph Partitioning* – Kernighan Lin

Para alcançar um bom desempenho em uma aplicação distribuída, é essencial que a aplicação apresente um bom balanceamento de carga e um bom controle da troca de mensagem entre processos, a fim de evitar gargalos que são prejudiciais no tempo de execução final da aplicação. O *InterSCSimulator* apresentado em [Santana et al. 2017] não apresenta um *speedup* escalável, pois a medida que se dobra a quantidade de processos mantendo o tamanho da entrada, o tempo não cai pela metade. Baseado no trabalho apresentado em [Bin Khunayn et al. 2017], que usou o algoritmo K-way para particionar os dados do SMARTS de [Ramamohanarao et al. 2017], este trabalho propõe o uso do algoritmo *Binary Graph Partitioning* – Kernighan Lin (BGP-KL) para solucionar ou mitigar esse problema apresentado em [Santana et al. 2017]. Em virtude de todos os atores (viagens) da simulação partirem ou estarem situados em um dado vértice do grafo, optou-se pela divisão do grafo e, consequentemente, dos dados atrelados aos vértices.

O BGP-KL pode ser dividido em dois passos: pré-processamento e particionamento do grafo.

2.1. Pré-processamento

O tempo para o particionamento do grafo G com a finalidade de reduzir o custo (peso de uma aresta que conecta dois grafos em partições diferentes) de conexão entre duas partições pode ser demasiadamente grande. Esse custo é devido ao número de comparações realizadas para encontrar o melhor corte, que estão diretamente relacionadas a quantidade de conexões (arestas) entre os vértices. Assim como em [Karypis and Kumar 1998] essa fase consiste em construir um novo grafo $Gi(Vi, Ei)$ menor a partir do grafo $G(V, E)$ inicial. Os vértices do grafo G original são combinados em um grupo de vértices Vi (formando um único vértice com a soma dos pesos de todos) como podemos ver demarcados por uma elipse vermelha no lado direito da Figura 1. Além disso, para preservar as relações de conectividade, as arestas de Vi são a união das arestas de seus vértices. Desse modo, é possível a redução do número de arestas de modo que Gi possa ser revertido ao G original após o particionamento.

2.2. Particionamento

Após o pré-processamento do grafo $G(V, E)$, o BGP-KL é aplicado no grafo $Gi(Vi, Ei)$. Ele particiona Gi com base no uso de uma estrutura de árvore binária para a decomposição da entrada entre os p processos. Em cada passo é aplicado o algoritmo de Kernighan Lin em Gi. O Kernighan apresenta uma complexidade de $O(|E|log|E|)$, porém, o seu custo pode ser reduzido para a complexidade de $O(|E|)$, [Karypis and Kumar 1998]. O KL divide o grafo em dois novos grafos minimizando o custo do corte. Em seguida, cada parte da partição é destinada para um nó da árvore binária que consequentemente realiza o mesmo processo recursivamente até que o número de nós folhas satisfaça um número de processos p definido *a priori* como mostra a Figura 1.

3. Experimento e Resultados

Para avaliar a solução foi testado o particionamento da entrada levando em consideração a distribuição dos vértices (esquinas e conexões entre ruas), arestas (ruas) e viagens. Para a realização desse experimento, foi utilizado um conjunto de dados da cidade de São Paulo

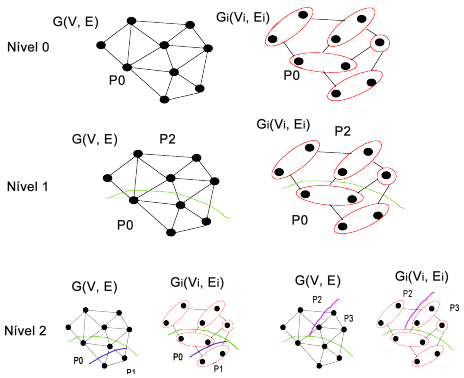

Figura 1. Processo de divisão do grafo entre quatro processos, $P0$, $P1$, $P2$ e $P3$.

com um total de 50.365 vértices, 118.905 arestas e 169.564 viagens partindo de diversos pontos da cidade. A máquina utilizada para realização do experimento possui 8 GB de memória RAM e processador Intel(R) Core(TM) it-7500U CPU 2.70 GHz com quatro núcleos.

O particionamento do grafo foi feito para 2, 4, 8 e 16 processos, porém, nessa seção apresentamos somente os resultados para 16 processos tendo em vista que o algoritmo apresenta o comportamento semelhante nos quatro casos. Para isso foi considerado no processo a divisão dos vértices (conexões entre ruas), arestas (ruas) e viagens (viagens de atores de um vértice a outro no grafo). Diante dos resultados obtidos mostrados na Tabela 1 pode se constatar que o número de vértices presente em cada processador é o mesmo, que o número de arestas é aproximadamente igual e que existem grandes discrepâncias nas partições quando se trata das viagens que foram alocadas de acordo com o seu vértice de origem.

Tabela 1. Particionamento dos vértices, arestas e viagens de um grafo Origem-Destino entre 16.

Processo	Vértices	Arestas	Viagens
0	3.147	14.198	8.545
1	3.148	14.360	6.736
2	3.148	14.387	15.156
3	3.148	14.807	14.537
4	3.147	14.529	9.194
5	3.148	14.324	7.405
6	3.148	14.579	7.679
7	3.148	14.625	23.015
8	3.147	14.394	13.312
9	3.148	14.678	7.756
10	3.148	14.716	8.612
11	3.148	14.700	12.289
12	3.148	14.402	9.539
13	3.148	14.416	9.553
14	3.148	14.430	7.783
15	3.148	14.698	8.453

4. Considerações Finais

Este trabalhou estudou a aplicação do algoritmo *Binary Graph Partitioning* – Kernighan Lin para o particionamento dos dados da entrada de um simulador de tráfego para cidades inteligentes, o *InterSCSimulator*. O objetivo do algoritmo é particionar os dados usados na simulação, com o intuito de obter-se um melhor balanceamento de carga ao longo da execução da simulação.

Nos resultados obtidos, pode-se notar que o algoritmo funcionou bem na distribuição do número de vértices e arestas. Porém, o mesmo não foi observado para o número de viagens, o que pode influenciar no balanceamento de carga no momento da execução. Assim, em trabalhos futuros pretende-se validar se o balanceamento de vértices e arestas obtidos é o suficiente ou se é necessário melhorar o particionamento das viagens, validando-o com a execução do *InterSCSimulator* em um ambiente distribuído. Além disso, pretende-se também avaliar como um grafo de dependência entre os atores da simulação assim como em [Bin Khunayn et al. 2017].

Referências

Bin Khunayn, E., Karunasekera, S., Xie, H., and Ramamohanarao, K. (2017). Exploiting data dependency to mitigate stragglers in distributed spatial simulation. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 43. ACM.

Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392.

Ramamohanarao, K., Xie, H., Kulik, L., Karunasekera, S., Tanin, E., Zhang, R., and Khunayn, E. B. (2017). Smarts: Scalable microscopic adaptive road traffic simulator. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):26.

Santana, E. F. Z., Lago, N., Kon, F., and Milojicic, D. S. (2017). InterSCSimulator: Large-scale traffic simulation in smart cities using erlang. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 211–227. Springer.