Avaliação e paralelização de algoritmos de mapeamento para grafos de visibilidade

Matheus Tavares, Isadora Cardoso, Vilker Tenorio
Tarciso Ferreira, Heitor S. Ramos

[1]LACCAN/Instituto de Computação – Universidade Federal de Alagoas (UFAL)
Av. Lourival Melo Mota, s/n – Tabuleiro do Martins – CEP 57072-900
Maceió – AL – Brasil

{math.tav,vilker.tenorio,isadoracardoso22,tarciso.fsf}@gmail.com

ramosh@acm.org

Resumo. *Neste artigo avaliamos o uso de técnicas de computação paralela para abordar o problema de análise de grandes volumes de dados oriundos de séries temporais. Utilizamos a representação de séries temporais em grafos de visibilidade, no qual cada elemento da série é representado por um vértice e as arestas representam visibilidade entre vértices, definida através de uma função de mapeamento. Para uma dada função de mapeamento, avaliamos dois algoritmos sequenciais e um terceiro algoritmo paralelo, feito na ferramenta Apache Spark, a fim de comparar o desempenho entre os mesmos. Os resultados são promissores e mostram que há melhoria no desempenho ao utilizarmos técnicas de computação paralela para o problema proposto. Como principal contribuição desse trabalho, conseguimos viabilizar a criação de grafos com* 10^6 *vértices utilizando um cluster com capacidade de* 0.291 *TFlops.*

1. Introdução

Séries temporais são uma sequência de valores ordenados observados em diferentes janelas de tempo. Essas séries são utilizadas em diversas áreas do conhecimento e, para sua análise, é interessante o uso de técnicas que possam extrair informações desses dados de maneira eficiente. Para isto, o mapeamento em grafos é uma técnica atrativa. No presente artigo empregou-se a técnica descrita em [Lacasa et al. 2008], chamada Grafo de Visibilidade (GV), que cita diversas propriedades de interesse, a saber: baixo custo computacional, implementação simples, capacidade de aplicação em todo tipo de série temporal, e, preserva a estrutura das séries temporais.

A medida que a série temporal cresce, o esforço computacional para manuseá-la também cresce. Dessa forma, a demanda por capacidade computacional para a obtenção de desempenho satisfatório pode se tornar impraticável para executar em apenas um computador. Há muitas formas de abordar computacionalmente este problema. No corrente estudo, foi utilizada a ferramenta *Apache Spark*, capaz de abstrair a complexidade presente em computação paralela de tal forma que usuários com pouco conhecimento na área consigam realizar análises sofisticadas com agilidade [Penchikala 2015].

2. Grafos de Visibilidade

O Grafo de Visibilidade (GV) é usado para mapear uma série temporal em um grafo, herdando muitas das suas propriedades e, dessa forma, é capaz de preservar estruturas

não-triviais acerca da série que representa.

Podemos descrever a técnica de visibilidade com a seguinte função de mapeamento: dois pares de valores arbitrários (t_a, y_a) e (t_b, y_b) serão visíveis entre si, e consequentemente serão nós conectados no grafo associado, se qualquer outro par (t_c, y_c) no qual $t_a < t_c < t_b$ satisfaz:

$$y_c < y_b + (y_a - y_b)\frac{t_b - t_c}{t_b - t_a}. \quad (1)$$

2.1. Propriedades do Grafo de Visibilidade

A principal vantagem da técnica adotada se refere a capacidade de incorporar a estrutura da série temporal à topologia do grafo associado, isto é, séries periódicas geram grafos regulares, séries aleatórias geram grafos aleatórios e séries fractais geram grafos livre de escala, assim, conservando suas características [Lacasa et al. 2008]. O GV correspondente a uma série temporal é necessariamente conexo pois todos os vértices se conectam ao menos aos seus vizinhos. Além disso, o GV é um grafo não-dirigido, uma vez que a forma como o algoritmo é construído não atribui direção às arestas.

Outra propriedade importante do GV é permanecer invariante a certas transformações na série temporal, como podemos ver na figura 1: (a) série temporal original, (b) translação, (c) redimensionamento vertical, (d) redimensionamento horizontal, (e) adição de uma linha de tendência. Notamos que o grafo de visibilidade gerado em todos esses casos é invariante.

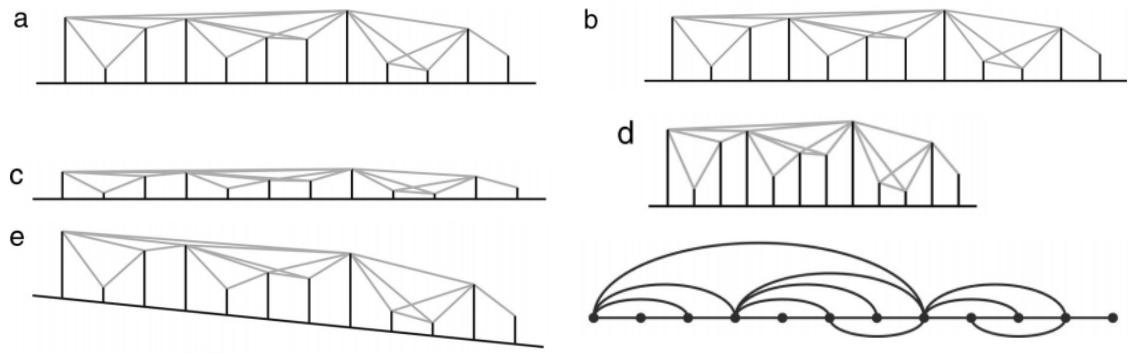

Figura 1. Transformações em uma Série Temporal e Grafo de Visibilidade [Lacasa et al. 2008].

3. Computação Paralela com Apache Spark

O *Apache Spark* [Holden Karau 2015] é uma poderosa ferramenta que tem como objetivo oferecer velocidade, facilidade de uso e análises sofisticadas para seus usuários, sem a necessidade de profundo conhecimento sobre sistemas distribuídos.

O *Apache Spark* [Spark 2015] conta com uma abstração de distribuição de sistema: o RDD (resilient distributed dataset), que é simplesmente uma coleção distribuída de forma automática de elementos. Toda operação com o *Spark* é realizada através de um RDD. O *Spark* permite duas operações: (a) Transformações (como o *map*), as quais

criam um novo conjunto de dados a partir de um já existente, e (b) Ações (como o foreach), as quais retornam um valor para o usuário após agir em cima do conjunto de dados. Todas as transformações no *Spark* são *lazy*, ou seja, os resultados não são computados imediatamente, apenas quando requisitados.

4. Problema

Analisar dados representados por séries temporais pode apresentar grandes demandas computacionais. No caso de grafos de visibilidade, até mesmo a construção do grafo com milhões de vértices pode ser impraticável. Esse problema pode dificultar ou até mesmo inviabilizar o uso de tal técnica para análises de grandes volumes de dados. Por essa razão, abordamos esse problema através da implementação de dois algoritmos sequenciais de mapeamento com diferentes complexidades e propomos um algoritmo paralelo. Para avaliação da nossa proposta, conduzimos um experimento comparando o desempenho entre eles.

Vale salientar que todos os algoritmos seguem os requisitos definidos por [Lacasa et al. 2008] e resultam em grafos isomorfos, ou seja, todos são igualmente válidos, tendo como diferença apenas o tempo e o processamento necessário para serem atingidos.

4.1. Algoritmos

É importante notar que em todos os algoritmos descritos a seguir os elementos da série se tornarão vértices do grafo e a visibilidade entre eles é representada pela existência de uma aresta que conecta os vértices.

4.1.1. Algoritmos Sequenciais

O algoritmo mais simples para a criação de GVs é apresentado a seguir: para cada elemento da série é gerado um vértice e automaticamente adiciona-se arestas entre os elementos vizinhos na série temporal. Em seguida, avalia-se a possível visibilidade do atual para todos os outros elementos subsequentes a ele na série através de uma função que, dado dois elementos - a e b - procura a existência de um elemento c que torne-os não visíveis. Se c não for encontrado, adiciona-se uma aresta entre a e b. Essa verificação é feita para todos os elementos da série temporal, o que torna sua complexidade $O(n^3)$. Para a avaliação da visibilidade entre os vértices, é utilizada a equação 1.

Notamos que o problema trabalhado pede no mínimo uma verificação de todos os vértices aos pares. Dessa forma, é desconhecida a obtenção de um algoritmo que possua uma complexidade menor que $O(n^2)$, (logo, é a solução ótima), como visto na equação 2.

$$\binom{n}{2} = \frac{n!}{(n-2)\,2!} = \frac{n(n-1)(n-2)!}{(n-2)!\,2} = \frac{n(n-1)}{2} = \frac{n^2-n}{2} \tag{2}$$

Pensando nisso, implementou-se um segundo algoritmo onde não percorre-se todos os elementos entre os pares da ação, pois a verificação de visibilidade ocorre durante a associação destes. Para isto, guarda-se o maior elemento da comparação anterior e utiliza-o, juntamente com os pares avaliando se o novo vértice está acima ou abaixo da

reta formada entre o vértice inicial e o elemento guardado anteriormente. Se a função retornar um valor menor que zero, então o esse novo vértice está acima da reta, logo, existe aresta entre os vértices analisados e o valor do maior elemento é atualizado; caso contrário, não é adicionada uma aresta entre os vértices.

4.1.2. Algoritmo Paralelo

Após a obtenção da solução ótima, torna-se possível paralelizar o problema para avaliação. No algoritmo a seguir, utilizamos funções próprias da linguagem *Scala*, que é suportada pelo *Spark*. A operação *SerieRDD.map* é executada em paralelo e aplica em cada elemento l da série o argumento em seu interior. A operação *serieList.foreach* realiza o teste de l com todos os elementos a sua direita na série, verificando se a condição de visibilidade é aceita, caso seja, adiciona o vértice visível à lista de arestas e atualiza o maior elemento.

Algorithm 1 FUNÇÃO DE VERIFICAÇÃO DE VISIBILIDADE EM PARALELO

function $Verifica\_Visibilidade\_Algorito\_O2\_Paralelo(a, b)$
$SerieRDD.map((l\_1, l\_2) =>$
$var\ arestas; var\ maiorElemento = 0; var\ valorMaiorElemento;$
$serie.foreach((a\_1, a\_2) =>$
if $(a\_1 > l\_1)$ **then**
if $maiorElemento == 0$ **then**
$arestas.add(a\_1);\ maiorElemento = a\_1;\ valorMaiorElemento = a\_2;$
else
if $Teste - de - visibilidade$ **then**
$arestas.add(a\_1);\ maiorElemento = a\_1;\ valorMaiorEelemento = a\_2$
end if
end if
end if
end function

5. Resultados

Para comparar o algoritmo paralelo implementado no *Spark* e os algoritmos sequenciais, realizou-se um estudo paramétrico de performance paralela para a simulação da função de mapeamento do GV. As métricas apresentadas referem-se a simulações realizadas na série temporal simulada, utilizando uma quantidade fixa de 14 processadores para o servidor *Spark* e variando a quantidade de elementos da série. A figura 2 apresenta o comportamento obtido para o tempo de processamento das simulações. Observa-se que o tempo de execução da paralelização diminui consideravelmente quando comparado com os algoritmos sequenciais. Observamos também que o algoritmo $O(n^3)$ não escala bem com o aumento da série. No caso dos algoritmos $O(n^2)$ e paralelo, ambos conseguiram manipular séries de 10^6 elementos, porém o algoritmo paralelo consegue construir o gráfico em um tempo em torno de 20% do necessário pelo algoritmo sequencial, apresentando um *speedup*, entre o algoritmo $O(n^2)$ e paralelo, de aproximadamente 5.

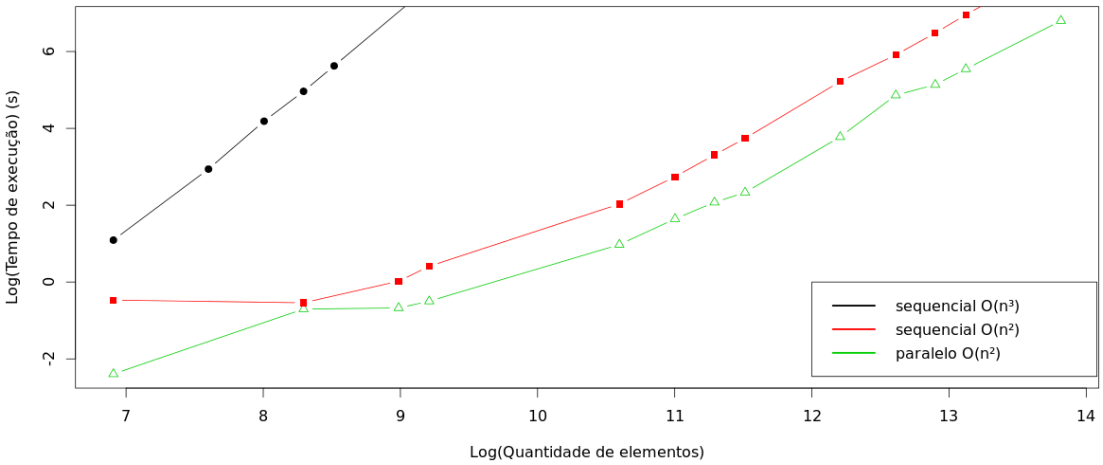

Figura 2. Tempo de execução entre os algoritmos propostos em função do tamanho da série

6. Conclusão

Através do estudo realizado, foi possível constatar que esta paralelização obteve melhores tempos de execução que os algoritmos sequenciais apresentados. Com as técnicas de computação paralela, um *cluster* de 14 máquinas foi capaz de gerar grafos de até 10^6 vértices em tempo hábil, o que torna a técnica de Grafos de Visibilidade para o mapeamento de séries temporais viável mesmo para grafos grandes.

7. References

Referências

Holden Karau, Andy Konwinski, P. W. M. Z. (2015). *Programming with RDDs*.

Lacasa, L., Luque, B., Ballesteros, F., Luque, J., and Nuño, J. C. (2008). From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*.

Penchikala, S. (2015). Big data processing with apache spark – part 1: Introduction.

Spark, A. (2015). Apache spark: Programming guide.