Análise de Dados Científicos: uma Análise Comparativa de Dados de Simulações Computacionais *

Thaylon Guedes[1], **Vítor Silva**[2], **José Camata**[2,3], **Marta Mattoso**[2], **Daniel de Oliveira**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (IC/UFF)

[2]COPPE - Universidade Federal do Rio de Janeiro

[3]Centro de Computação de Alto Desempenho e Departamento de Engenharia Civil

{thaylongs, danielcmo}@ic.uff.br, {silva, marta}@cos.ufrj.br, camata@nacad.ufrj.br

Resumo. *Os avanços nas simulações computacionais têm permitido o processamento de volumes de dados cada vez maiores. Para representar as estruturas de dados complexas inerentes de tais simulações, elas são armazenadas em arquivos de formatos heterogêneos. Carregar tais dados em um SGBD, como o SciDB, para apoiar as análises deles se torna uma tarefa complexa, ou mesmo inviável, devido ao seu volume e/ou estrutura. Para evitar esse carregamento, existem abordagens que realizam consultas adaptativas e/ou que indexam os arquivos. Escolher a mais adequada pode não ser trivial. Neste artigo realizamos uma análise comparativa em termos de desempenho das abordagens de consulta de dados produzidos por uma simulação em dinâmica de fluídos computacional.*

1. Introdução

Diante dos avanços nas simulações computacionais, o volume de dados produzidos pelos programas de simulação tem aumentado cada vez mais. Esses dados representam resultados parciais e finais, sendo comumente armazenados em arquivos, que assumem diferentes formatos de acordo com o domínio da aplicação (*e.g.*, XDMF e HDF5). Neste artigo, nomeamos os dados representados em sua forma bruta simplesmente como dados científicos. Nesse cenário, para apoiar a análise de dados científicos, os conteúdos dos arquivos precisam ser acessados, extraídos e indexados, a fim de que sejam carregados em um repositório externo ou uma base de dados para permitir a análise. Existem diversas abordagens que lidam com o acesso, a extração e a indexação de dados científicos envolvidos em simulações computacionais, mantendo esses dados em seu formato original [Blanas et al. 2014]. Essas abordagens comumente acessam dados específicos de domínio em arquivos por meio das análises léxica e sintática, depois extraem esses dados de acordo os atributos de interesse do usuário por meio de um catálogo de dados (específico para cada formato de arquivo de acordo com o domínio da simulação) e indexam esses dados usando linguagens específicas de consulta e APIs. A indexação dos dados científicos em seu formato original permite que o volume de dados carregados em um repositório externo seja reduzido, e, consequentemente, a sobrecarga em termos de tempo também deve ser reduzida. Ademais, essas soluções aliviam o trabalho dos usuários quanto ao desenvolvimento de seus próprios códigos para cada tipo de análise, ao propor recursos para o processamento de consultas. Como exemplos, temos o FastBit [1], o FastQuery [2] e o NoDB [3].

*Este artigo foi financiado parcialmente pelo CNPq, CAPES e FAPERJ

[1]https://sdm.lbl.gov/fastbit

[2]http://www-vis.lbl.gov/Events/SC05/HDF5FastQuery

[3]http://dias.epfl.ch/nodb

Entretanto, essas soluções são baseadas na gerência de dados em arquivos isolados (*i.e.* um arquivo não tem relação com outros produzidos) [Silva et al. 2017, Karpathiotakis et al. 2014]. Porém, em simulações computacionais, é produzida uma série de arquivos que possuem associação entre si. Para gerenciar os dados envolvidos ao longo das simulações, registros do histórico dos dados consumidos e produzidos pelos programas de simulação (*i.e.*, dados de proveniência [Freire et al. 2008]) são capturados. Ademais, os elementos de dados (conteúdo dos arquivos) capturados precisam que seus relacionamentos (ou dependências de dados) com elementos de dados obtidos de outros arquivos sejam representados [Silva et al. 2017]. Logo, a análise exploratória de dados científicos em simulações computacionais envolve tanto a gerência dos dados científicos, como dos dados de proveniência. Recentemente, foi proposto o modelo PROV-Df [Silva et al. 2017], que segue o padrão definido pelo W3C PROV [Moreau and Groth 2013], para apoiar a gerência do fluxo de arquivos e de elementos de dados ao longo da execução de simulações, integrando dados de proveniência, de execução, e científicos.

Este artigo considera a captura e a indexação de dados científicos a partir de arquivos, assim como a carga dos índices gerados em uma base de dados de acordo com o modelo PROV-Df. Mais especificamente, este artigo propõe uma análise comparativa de técnicas de indexação de dados científicos usando 3 abordagens: FastBit, NoDB e índice posicional. Como *baseline*, comparamos as técnicas de indexação com a carga de todos os elementos de dados (*i.e.*, valores dos atributos de interesse extraídos) em uma base de dados relacional (usando PostgreSQL). Além desta introdução, este artigo contém outras 4 seções. A seção 2 apresenta abordagens existentes para a indexação de dados científicos. A seção 3 discute o estudo de caso baseado em uma simulação em dinâmica de fluidos computacional, assim como o volume e a natureza dos dados manipulados. A seção 4 apresenta a análise comparativa realizada. Por último, a seção 5 conclui esse trabalho e apresenta os trabalhos futuros.

2. Abordagens Existentes para a Indexação de Dados Científicos

Existem diferentes técnicas baseadas na indexação de dados científicos, sendo que duas propostas de índices têm sido largamente utilizadas: mapa de *bits* (*bitmap*) e posicional. A indexação baseada em um mapa de *bits* consiste na análise do domínio de determinados atributos presentes em dados científicos para gerar índices booleanos por meio da verificação de equações ou inequações algébricas. Por exemplo, assumindo-se que os valores do atributo *X* para um arquivo de dados científicos apresentam um domínio com apenas 5 valores possíveis na estrutura de dados de inteiro, então o mapa de *bits* necessita de cinco colunas para identificar a presença ou não de um valor desse atributo por meio de uma equação algébrica. Existem casos também em que o uso de inequações para a indexação *bitmap* é mais vantajoso, principalmente em cenários em que o domínio de um determinado atributo é muito extenso. O FastBit e o FastQuery são exemplos de sistemas que realizam a indexação de dados em mapas de *bits*. Mais especificamente, o FastBit propõe um conjunto de variações nas técnica de indexação, permitindo o ajuste do tipo de codificação utilizado; do algoritmo de compressão; e do nível de precisão dos valores numéricos para realizar o mapa de *bits*.

Outra técnica de indexação de dados científicos considera a geração de índices posicionais, sendo que estes utilizam informações que facilitam a localização (*i.e.*, posição) dos dados científicos em arquivos. Em comparação à indexação *bitmap*, os índices posicionais podem fornecer uma sobrecarga de dados menor, pois estes não utilizam *bits* redundantes (*i.e.*, sequência de zeros no mapa de *bits*) que crescem de acordo com o domínio dos atributos indexados. Além disso, o índice posicional permite referenciar estruturas

de dados mais complexas, como árvores e malhas. O NoDB [Karpathiotakis et al. 2014] é um exemplo de sistema que emprega índices posicionais e realiza a extração de dados de domínio presentes em arquivos, carregando-os em uma versão modificada do SGBD PostgreSQL, conhecido como PostgresRaw. Apesar de evitar mudanças nas estruturas de dados adotadas pelos arquivos, o NoDB baseia-se no processamento de consultas adaptativas, que utiliza estatísticas e estratégias de *caching* para ter acesso aos dados científicos e realizar as transformações de dados estritamente necessárias. No NoDB apenas os dados científicos necessários pelas consultas são efetivamente indexados. Entretanto, como limitação, o NoDB realiza a carga dos dados científicos acessados no PostgresRaw, o que caracteriza uma sobrecarga de armazenamento dos dados.

Entretanto, os trabalhos discutidos focam apenas em arquivos "isolados", ou seja, eles não permitem análises sobre o fluxo de dados. Para que o fluxo fosse representado nos SGBDs adaptados aos dados científicos, os programas de simulação que geram os dados teriam que ser reescritos, mudando todo o seu acesso às estruturas de dados para o acesso ao SGBD, algo que não seria viável no panorama atual de programas de simulação computacional. A outra alternativa seria manter os dados em seus formatos originais e replicá-los nos sistemas de gerência e análise de dados científicos, algo que não se mostra viável com o NoDB. Dessa forma, conforme mencionado anteriormente, os trabalhos existentes não permitem ainda a gerência dos diversos arquivos envolvidos e dos elementos de dados relacionados pelos programas que compõem a simulação computacional.

3. Estudo de Caso: Dinâmica de Fluidos Computacional

O estudo de caso desse artigo é uma simulação no domínio de dinâmica de fluidos computacional, focada em um problema de análise da deposição de sedimentos em bacias sedimentares. Esse estudo de caso foi utilizado pelo fato de envolver a geração de um grande volume de dados científicos armazenados em formatos específicos do domínio (XDMF e HDF5), sendo que os usuários necessitam do conteúdo desses arquivos para apoiar as suas análises (validação de hipóteses científicas). Especificamente, os experimentos realizados utilizaram o resolvedor libMesh-sedimentation [Silva et al. 2016], que consiste em uma aplicação construída a partir da biblioteca libMesh [4] para simular correntes turbidíticas, tipicamente encontradas em processos geológicos. Nesse caso, os sedimentos são transportados devido à movimentação do fluido, e são representados por meio de equações de Navier-Stokes de fluido incompressível combinado com uma equação de transporte dominada por advecção (concentração de sedimentos) [Guerra et al. 2016].

Nessa simulação computacional, os resultados parciais e finais das malhas são frequentemente armazenadas em arquivos de dados científicos no formato XDMF e HDF5, enquanto outros dados de interesse são mantidos em estruturas de dados em memória (*i.e.*, como variáveis no código do resolvedor). Entretanto, para capturar os dados em memória, o libMesh-sedimentation foi integrado ao ParaView Catalyst [Ayachit et al. 2015] para capturar esses dados científicos da malha simulada pelo resolvedor e armazená-los em arquivos no formato CSV. Esses arquivos no formato CSV apresentam informações como a direção de movimentação do fluido, a concentração de sedimentos em suspensão e depositados, as coordenadas do ponto na malha, entre outras.

A Figura 1 apresenta as transformações de dados envolvidas no libMesh-sedimentation, na cor laranja, assim como as transformações de dados usando o ParaView Catalyst, na cor azul, que envolveram a geração de dados científicos capturados nesse experimento. Na

[4] http://libmesh.github.io

análise comparativa desse artigo, considerou-se o processo de acesso, extração e indexação de dados envolvidos após a geração dos arquivos no formato CSV pelas transformações.

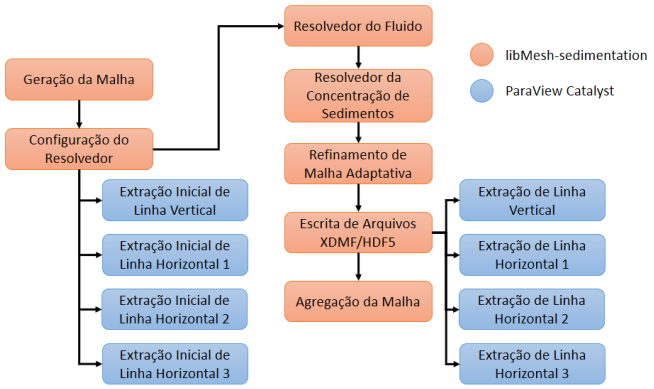

Figura 1. Simulação computacional usando libMesh-sedimentation e ParaView Catalyst

4. Análise Comparativa

Neste trabalho foram analisadas 3 abordagens de indexação: FastBit, PostgresRaw (NoDB) e índice posicional. Esta última abordagem armazena a posição do registro de acordo com atributos presentes nos arquivos de dados científicos, sendo que o seu código foi baseado na solução de indexação do NoDB. A simulação computacional do CFD foi executada no *cluster* Lobo Carneiro[5], usando 480 *cores* e produzindo aproximadamente 47 GB de dados em 97.163 arquivos. Para que seja possível realizar a indexação, os dados científicos de interesse precisaram ser acessados e extraídos dos arquivos gerados. A etapa de extração nesse experimento obteve 4.65 MB de dados presentes nos arquivos XDMF e HDF5 em 235 segundos. Uma análise comparativa dessas abordagens de indexação de dados é realizada na Seção 4.1.

Essa análise comparativa foi realizada em uma máquina local que possui um processador Intel Core i7-7700HQ com 2.80GHz e 8 GB de memória RAM. Sua unidade de armazenamento consiste em um SSD com até 1.570 MB/s de leitura e 540 MB/s de escrita. O Ubuntu 14.04.02 LTS foi utilizado como sistema operacional com o FastBit na versão 2.0.3 e o PostgresRaw compilado de acordo com a última versão disponível no repositório[6]. Em relação ao último *commit* desse repositório (do dia 06/02/2017), modificou-se apenas a quantidade máxima de arquivos de dados científicos a serem indexados. Enquanto o valor padrão desse parâmetro era 50, nossa aplicação considerou a indexação de 808 arquivos.

4.1. Comparação das Diferentes Abordagens de Indexação

Entre as abordagens escolhidas, somente o FastBit dispõe de parâmetros de configuração, sendo os principais o *binning* e o *encoding*. O *binning* determina como o *bitmap* é produzido, e no caso foi utilizado o seu valor padrão. Enquanto o *encoding* manipula os mapas de *bits* para ocupar menos espaço de armazenamento em disco, sem perda de informação, ou otimizar o processamento de consultas. Neste último parâmetro utilizamos como argumento o "*interval-equality*", pois ele é recomendado para o tipo de dados científicos extraídos e indexados neste artigo, os quais apresentaram alta cardinalidade. Logo, nesse caso, favoreceu-se o desempenho no processamento de consultas, ao invés do espaço de armazenamento em disco.

[5]http://www.nacad.ufrj.br/pt/recursos/sgiicex
[6]https://github.com/HBPMedical/PostgresRAW

O primeiro item a ser comparado é o tempo necessário para realizar a indexação dos dados. A Figura 2 apresenta o tempo para indexar dados científicos extraídos de arquivos no formato XDMF e HDF5, utilizando as abordagens PostgresRaw, índice posicional e FastBit. Nesse caso, o PostgresRaw possui o menor tempo de indexação dos dados científicos, pois, ao acessar esses dados, essa ferramenta já tira proveito dos dados em memória para realizar a indexação dos mesmos. Apesar de apresentar o mesmo comportamento (ou seja, geração de índices posicionais) que a abordagem de índice posicional, o PostgresRaw difere ao apresentar uma política de procrastinar a indexação de dados. Assim, o PostgresRaw utiliza uma abordagem adaptativa, em que os índices só são gerados, quando solicitados pelas consultas submetidas. Consequentemente, nas primeiras consultas submetidas para acessar os dados científicos em um arquivo, espera-se que o PostgresRaw apresente um tempo de processamento das consultas maior, pois os índices posicionais ainda serão gerados.

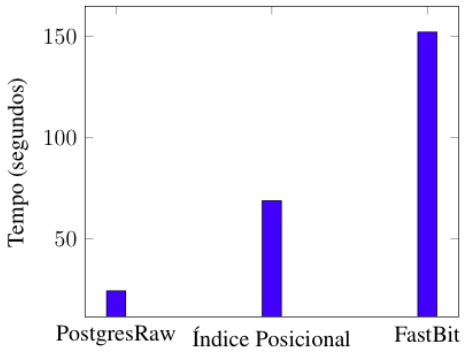

Figura 2. Comparação do tempo de indexação para as três abordagens.

Ainda na Figura 2 pode ser observado que o tempo de execução do FastBit é mais que o dobro do índice posicional. Entretanto, o FastBit responde as consultas com intervalos muito mais rápido do que no índice posicional, uma vez que, no índice posicional, o processamento de consultas requer a varredura de cada linha e coluna, assim como a conversão dos índices para os valores dos atributos (*i.e.*, dados científicos), de acordo com a estrutura de dados mais adequada. Por outro lado, o FastBit apresenta um desempenho melhor para realizar consultas em função da operação binária de *AND* no mapa de *bits* de acordo com as colunas que devem ser analisadas ou investigadas.

No quesito de armazenamento em disco, os algoritmos não mantiveram o comportamento apresentado na Figura 2. Na Tabela 1 é apresentado o espaço de armazenamento em disco necessário para amazenar os índices gerados, em *megabytes*, por cada abordagem de indexação. Como pode ser observado, o PostgresRaw foi o que mais necessitou de espaço em disco. Isso pode ser consequência dos índices verticais e posicionais criados na fase de processamento.

Método	PostgresRaw	Índice Posicional	FastBit
Espaço em disco (MB)	26,30	7,96	18,30

Tabela 1. Espaço em disco utilizado para a indexação de dados

Ao realizar a comparação entre o FastBit e o Índice Posicional, observa-se que a velocidade de acesso é proporcional ao espaço em disco necessário para representar os índices. No FastBit, utiliza-se apenas um *bit* para determinar a existência de um determinado dado científico de interesse em uma linha da malha gerada pela aplicação de dinâmica

de fluidos, enquanto as outras colunas do *bitmap* apresentam seus valores zerados. Esse comportamento é conhecido por redundância de zeros. Consequentemente, é necessário mais espaço de armazenamento em disco para representar tal *bitmap*. Considerando-se o índice posicional, essa abordagem exige mais *bits* para descrever a posição dos dados científicos no arquivo, além do tamanho, em número de *bits*, do espaço ocupado no arquivo para armazenar esse dado.

5. Conclusão e Trabalhos Futuros

Devido ao grande volume de dados manipulados pelos programas em simulações computacionais reais, se torna inviável carregar todos os dados em um SGBD. Logo, tais dados devem ser indexados em seu formato original para que sejam passíveis de consulta. Nos experimentos deste artigo, apresentou-se uma análise dos tempos de execução das técnicas de indexação, sendo que o PostgresRaw apresentou o menor tempo de execução em relação ao FastBit e ao índice posicional. Por outro lado, essa técnica foi a que apresentou o maior custo de armazenamento em disco (26,30 MB), aproximadamente 3,3 vezes maior que o volume de dados do índice posicional (7,96 MB), que obteve o melhor resultado. Como trabalhos futuros, pretende-se executar essa simulação computacional usando um caso real mais complexo. Além disso, pretende-se analisar o desempenho do processamento de consultas usando diferentes abordagens de indexação de dados científicos. Outra proposta de trabalho futuro consiste no uso de soluções de indexação de dados científicos que tirem proveito do paralelismo, como o DiNoDB e o FastQuery.

Referências

Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., and Mauldin, J. (2015). Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV2015, pages 25–29, New York, NY, USA. ACM.

Blanas, S., Wu, K., Byna, S., Dong, B., and Shoshani, A. (2014). Parallel data analysis directly on scientific file formats. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 385–396, New York, NY, USA. ACM.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Computing in Science Engineering*, 10(3):11–21.

Guerra, G. M., Zio, S., Camata, J. J., Dias, J., Elias, R. N., Mattoso, M., B. Paraizo, P. L., G. A. Coutinho, A. L., and Rochinha, F. A. (2016). Uncertainty quantification in numerical simulation of particle-laden flows. *Computational Geosciences*, 20(1):265–281.

Karpathiotakis, M., Branco, M., Alagiannis, I., and Ailamaki, A. (2014). Adaptive query processing on raw data. *Proc. VLDB Endow.*, 7(12):1119–1130.

Moreau, L. and Groth, P. T. (2013). *Provenance: An Introduction to PROV*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers.

Silva, V., Camata, J., de Oliveira, D., Coutinho, A. L., Valduriez, P., and Mattoso, M. (2016). In situ data steering on sedimentation simulation with provenance data. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'16.

Silva, V., Leite, J., Camata, J. J., de Oliveira, D., Coutinho, A. L., Valduriez, P., and Mattoso, M. (2017). Raw data queries during data-intensive parallel workflow execution. *Future Generation Computer Systems*, 75:402 – 422.