Suporte ao Processamento Paralelo e Distribuído em uma DSL para Visualização de Dados Geoespaciais

Conference Paper · October 2018

CITATIONS
0

READS
60

7 authors, including:

Adriano Vogel
Pontifícia Universidade Católica do Rio Grande do Sul
33 PUBLICATIONS **69** CITATIONS
SEE PROFILE

Cassiano Rista
Pontifícia Universidade Católica do Rio Grande do Sul
9 PUBLICATIONS **17** CITATIONS
SEE PROFILE

Dalvan Griebler
Pontifícia Universidade Católica do Rio Grande do Sul
92 PUBLICATIONS **211** CITATIONS
SEE PROFILE

Isabel Harb Manssour
PUCRS - Pontifícia Universidade Católica do Rio Grande do Sul
76 PUBLICATIONS **122** CITATIONS
SEE PROFILE

Suporte ao Processamento Paralelo e Distribuído em uma DSL para Visualização de Dados Geoespaciais

Endrius Ewald, Adriano Vogel, Cassiano Rista, Dalvan Griebler, Isabel Manssour, Luiz Gustavo Fernandes

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

Email:adriano.vogel@acad.pucrs.br

Abstract. *The amount of data generated worldwide related to geolocalization has exponentially increased. However, the fast processing of this amount of data is a challenge from the programming perspective, and many available solutions require learning a variety of tools and programming languages. This paper introduces the support for parallel and distributed processing in a DSL for Geospatial Data Visualization to speed up the data pre-processing phase. The results have shown the MPI version with dynamic data distribution performing better under medium and large data set files, while MPI-I/O version achieved the best performance with small data set files.*

Resumo. *Tecnologias de geolocalização tem gerado uma grande quantidade de dados. No entanto, o processamento desse volume de dados não é uma tarefa trivial, pois os métodos de geolocalização exigem a aprendizagem de várias ferramentas ou linguagens de programação. Esse artigo apresenta o suporte ao processamento paralelo e distribuído para uma DSL de visualização de grandes volumes de dados geoespaciais. O sistema aproveita o processamento distribuído para acelerar o pré-processamento de dados. A versão com distribuição dinâmica dos dados (MPI-D) apresentou os melhores resultados com arquivos de dados grandes e médios. Enquanto isso, com arquivos de dados pequenos, o desempenho foi superior na versão MPI-I/O.*

1. Introdução

Nos últimos anos, ocorreu um crescimento significativo no volume de dados digitais gerados em todo o planeta. Em dezembro de 2012, um estudo realizado pelo IDC (*International Data Corporation*) [IDC 2012] estimou o tamanho do universo digital em cerca de 2.837 exabytes e com previsão de crescimento para incríveis 40,000 exabytes até 2020. Então, em 2020, de acordo com o IDC, o universo digital seria capaz de disponibilizar mais de 5 terabytes para cada pessoa do planeta. Esse estudo demonstra que estamos vivendo na era dos dados, ou seja, a era do *big data*.

Essa situação tem gerado demandas por novas técnicas e ferramentas que transformem os dados armazenados e processados em conhecimento. Nesse contexto, o uso e o aprimoramento das técnicas de visualização de dados geoespaciais surgem como uma alternativa. As técnicas de visualização usualmente são aplicadas à informações geoespaciais, usando Sistemas de Informação Geográfica (SIG), bibliotecas de programação e *frameworks*. As técnicas de visualização de dados geoespaciais permitem a visualização de variáveis associadas a uma localização espacial, tais como a população e o índice de

qualidade de vida de diferentes cidades, ou as vendas de uma empresa em uma região. Entre as técnicas de análise de dados geoespaciais, a visualização de dados se destaca por auxiliar os usuários a obterem informações rapidamente [Kraak and Ormeling 2011].

É importante destacar que as ferramentas disponíveis atualmente não fornecem as abstrações necessárias para a etapa de pré-processamento do *pipeline* de visualização [Moreland 2013]. Assim, mesmo que a visualização de dados ofereçam muitos benefícios, sua geração continua sendo um desafio [Zhang et al. 2015]. Os usuários têm dificuldades em lidar com a grande quantidade de dados, uma vez que exige custos elevados de processamento e um grande esforço de programação para manipular os dados brutos e o suporte ao paralelismo.

O pré-processamento tem um papel fundamental para a verificação de dados e erros, identificação de inconsistências e possíveis incompletudes. Na literatura (Seção 2), o pré-processamento de grandes quantidades de dados é considerado um problema de desempenho e citado em diferentes trabalhos. Dessa forma, o uso de arquiteturas de processamento paralelo e distribuído aparecem como uma alternativa capaz de atenuar esse problema. Isso é possível, graças ao uso de técnicas de programação paralela, que permitem a aceleração do processamento para as aplicações através dessas arquiteturas de alto desempenho. No entanto, esta não é uma tarefa trivial, pois requer habilidades específicas em ferramentas, metodologias e modelagem [Rünger and Rauber 2013].

Este trabalho avalia estratégias para o pré-processamento distribuído, buscando melhorar a experiência de criação de visualizações de dados geoespaciais, reduzindo o tempo necessário para o pré-processamento dos dados e a implementação da visualização de dados. O objetivo é permitir o processamento paralelo e distribuído na DSL GMaVis, que até o presente momento, era capaz de realizar apenas o processamento paralelo em sistemas multi-core [Ledur et al. 2017]. Assim, o trabalho apresenta as seguintes contribuições:

- Um estudo e implementação do processamento paralelo e distribuído para DSL GMaVis.
- Um conjunto de experimentos com diferentes implementações distribuídas do pré-processamento e tamanhos de arquivos baseados em cargas realísticas.

O artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados e a Seção 3 apresenta a linguagem da DSL GMaVis. A Seção 4 descreve a implementação do suporte ao pré-processamento distribuído. Os experimentos, resultados e discussões são mostrados na Seção 5. As conclusões deste trabalho são apresentadas na Seção 6.

2. Trabalhos Relacionados

Nessa seção são apresentadas diferentes abordagens de processamento paralelo e distribuído e DSLs para a visualização e pré-processamento de grandes quantidades de dados para aplicações de geovisualização, incluindo também abordagens para o aprimoramento de desempenho com base nas características de Entrada ou Saída (E/S).

No trabalho de [Seo et al. 2015], foram abordadas operações de E/S coletivas sem bloqueio em MPI. A implementação em MPICH foi baseada no algoritmo coletivo de

E/S ROMIO, substituindo as operações de bloqueio coletivo de E/S ou contrapartes não-bloqueantes. Os resultados indicaram um melhor desempenho que o bloqueio de E/S coletivo em termos de largura de banda de E/S, sendo capaz de sobrepor E/S com outras operações. [Latham et al. 2017] propõem uma extensão para o MPI-3, permitindo determinar quais nós do sistema compartilham recursos comuns. A implementação realizada em MPICH fornece um mecanismo portátil para a descoberta de recursos, possibilitando determinar quais nós compartilham dispositivos locais mais rápidos. Os resultados obtidos com testes de *benchmarks* demonstraram a eficiência da abordagem para investigar a topologia de um sistema. [Mendez et al. 2017] apresentam uma metodologia para avaliar o desempenho de aplicações paralelas com base nas características de E/S da aplicação, requisitos e níveis de severidade. A implementação definiu o uso de cinco níveis de severidade considerando requisitos de E/S de aplicações paralelas e parâmetros do sistema de HPC. Resultados mostraram que a metodologia permite identificar se uma aplicação paralela é limitada pelo subsistema de E/S e identificar possíveis causas do problema.

[Ayachit 2015] descreve o projeto e as características do *ParaView*, que é uma ferramenta de código aberto multiplataforma que permite a visualização e análise de dados. No *ParaView* a manipulação de dados pode ser feita interativamente em 3D ou através de processamento em lote. A ferramenta foi desenvolvida para analisar grandes conjuntos de dados usando recursos de computação de memória distribuída. [Wylie and Baumes 2009] por sua vez, apresentam um projeto de expansão para a ferramenta de código aberto *Visualization Toolkit* (VTK). O projeto foi denominado *Titan*, e oferece suporte a inserção, processamento e visualização de dados. Além disso, a distribuição de dados, o processamento paralelo e a característica cliente/servidor da ferramenta VTK fornecem uma plataforma escalável.

[Steed et al. 2013] descrevem um sistema de análise visual, chamado Ambiente de Análise de Dados Exploratório (EDEN), com aplicação específica para análise de grandes conjuntos de dados (*Big Data*) inerentes à ciência do clima. EDEN foi desenvolvido como ferramenta de análise visual interativa permitindo transformar dados em *insights*, melhorando assim a compreensão crítica dos processos do sistema terrestre. Resultados foram obtidos com base em estudos do mundo real usando conjuntos de pontos e simulações globais do modelo terrestre (CLM4). [Perrot et al. 2015] apresentam uma arquitetura para aplicações de *Big Data* que permite a visualização interativa de mapas de calor em larga escala. A implementação feita em *Hadoop*, *HBase*, *Spark* e *WebGL* inclui um algoritmo distribuído para calcular um agrupamento de *canopy*. Os resultados comprovam a eficiência da abordagem em termos de escalabilidade horizontal e qualidade da visualização produzida.

O estudo apresentado por [Zhang et al. 2015] exploram o uso de tecnologias de análise geovisuais e computação paralela para problemas de otimização geoespacial. O desenvolvimento resultou em um conjunto de ferramentas geovisuais interativas capazes de direcionar dinamicamente a busca por otimização de forma interativa. Os experimentos revelam que a análise visual eficiente e a busca através do uso de árvores paralelas são ferramentas promissoras para a modelagem da alocação do uso da terra.

Nesta seção foram apresentadas diferentes abordagens relacionadas com o pré-processamento e aprimoramento de desempenho de grandes quantidades de dados. Algumas abordagens [Seo et al. 2015], [Latham et al. 2017] são focadas no aprimoramento de

desempenho de aplicações de E/S, enquanto outras [Mendez et al. 2017] permitem identificar se a aplicação é limitada pelo subsistema de E/S. Algumas abordagens [Ayachit 2015], [Wylie and Baumes 2009] estavam preocupadas com a visualização e análise dos conjuntos de dados e outras em permitir a visualização interativa de aplicações *Big Data* [Steed et al. 2013], [Perrot et al. 2015]. Por fim, [Zhang et al. 2015] demonstram o uso de tecnologias de análise geovisuais através de árvores paralelas.

É possível observar que a literatura não apresenta estudos que otimizam o pré-processamento de grandes quantidades de dados levando em consideração uma DSL para um ambiente de processamento paralelo e distribuído. Além dessa lacuna observada, nesse estudo a GMaVis é estendida para suportar pré-processamento distribuído de dados. Ainda, diferentes implementações e tamanhos de arquivos são testados nesse estudo.

3. GMaVis DSL

GMaVis [Ledur et al. 2017] é uma DSL que fornece uma linguagem de especificação de alto nível e tem como objetivo simplificar a criação de visualizações para dados geoespaciais em larga escala. A DSL permite aos usuários filtrar, classificar, formatar e especificar a visualização dos dados. Além disso, a GMaVis tem expressividade específica para reduzir a complexidade e automatizar decisões e operações, tais como o pré-processamento de dados, o zoom e a localização do ponto inicial [Ledur et al. 2015].

```
1 visualization: clusteredmap;
2 settings {
3   latitude: field 7;
4   longitude: field 8;
5   marker-text: field 1 field 2;
6   page-title: "Airports in World";
7   size: full;
8 }
9 data {
10  file: "vis_codes/airports.data";
11  structure {
12    delimiter: ',';
13    end-register: newline;
14  }
15 }
```

Listagem 1. Código da DSL GMaVis para visualizar dados em *clusters*.

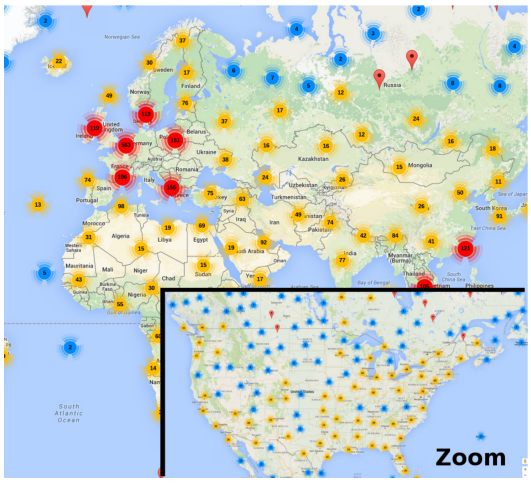


Figura 1. Visualização dos dados dos aeroportos pelo mundo.

A Listagem 1 ilustra um exemplo da linguagem de alto nível da GMaVis, e a visualização resultante é apresentada na Figura 1. A primeira declaração na Linha 1 é um *visualization*: declaração, na qual o tipo de visualização escolhido para aparecer na visualização é o *clusteredmap*. Na Linha 2, o bloco *settings* começa com as declarações usadas para especificar detalhes dos aspectos visuais e recebe os campos onde os atributos importantes estão localizados. Por exemplo, a *latitude* e a *longitude* estão sendo declaradas nas Linhas 3 e 4, quando são informados seus valores, que correspondem à posição no conjunto de dados onde essas informações podem ser encontradas. A Linha 5 possui um *marker-text* para ser exibido como um texto no marcador quando o usuário clica nele. A declaração *page-title* na Linha 6 informa o título que será colocado na visualização. Na

Linha 7, uma declaração *size* é usada para definir o tamanho que a visualização ocupará na página Web. A declaração permite também valores como *small* ou *medium*.

Há também um bloco *data* com declarações entre as Linhas 9 a 15 que especificam os arquivos e filtros de entrada. Os usuários também podem incluir sub-blocos para a estrutura e classificação dos dados. O arquivo de entrada é declarado na Linha 10, recebendo uma *string* com o caminho do sistema para o arquivo. Esta declaração pode ser repetida várias vezes até que seja incluído todo o conjunto de dados. Além disso, um sub-bloco *structure* é declarado na linha 11, com um *delimiter* e uma declaração *end-register* especificando que a vírgula separa os valores do conjunto de dados de entrada e o caractere de nova linha separa os registros. Esta declaração pode receber qualquer caractere ou palavras-chave definidas, tais como: *tab*, *comma*, *semicolon* ou *newline*.

3.1. Pré-processamento de Dados

O modulo de pré-processamento de dados é responsável pela transformação dos dados de entrada, aplicando operações de filtragem e classificação. Este módulo permite à DSL abstrair a primeira fase do *pipeline* para a criação da visualização [Moreland 2013], evitando que os usuários tenham que tratar manualmente com grandes conjuntos de dados. O módulo funciona recebendo os dados de entrada, processando e salvando em um arquivo de saída os dados formatados e estruturados. As principais operações são *Read*, *Process* e *Write*, sendo definidas juntamente com outras operações na Tabela 1.

Tabela 1. Operações de pré-processamento de dados e suas definições.

Definição	Descrição
$F = \{\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n\}$	F é um conjunto de arquivos de entrada a serem processados e α representa um único arquivo de um conjunto de dados particionado.
$Split(\alpha)$	Divide um arquivo do conjunto de dados de F em N blocos.
$D = \{d_1, d_2, d_3, ..., d_n\}$	D é um conjunto de blocos de um único arquivo. Pode-se dizer que D é o resultado de uma função $Split(\alpha)$.
$Process(D)$	Processa um único arquivo D de F.
$Read(d)$	Abre e lê um bloco de dados d de um α em F.
$Filter(d)$	Filtra um determinado bloco de dados d em D, produzindo um conjunto de registros para criar a visualização.
$Classify(...)$	Classifica os resultados de $Filter(...)$.
$Write(...)$	Salva os resultado de $\sum_{i=1}^{n} Process(F)$, onde F representa um conjunto de arquivos (α) em um arquivo de saída a ser usado na geração da visualização.

O compilador da DSL usa detalhes do código-fonte para gerar esse pré-processamento de dados através da linguagem de programação C++ [Ledur et al. 2017]. A escolha do C++, é justificada pelo fato de permitir a criação de uma aplicação usando um vasto conjunto de interfaces de programação paralela, além de possibilitar aprimoramentos de baixo nível no gerenciamento de memória e leitura de disco. Assim, o compilador gera um arquivo chamado *data_preprocessor.cpp*. Sendo todo código gerado e executado sequencialmente, incluindo bibliotecas, constantes e tipos de dados. A execução da aplicação de geovisualização usando a GMaVis pode ser paralela usando o suporte ao paralelismo em ambientes multi-core oferecido pela DSL SPar [Griebler et al. 2017]. Além

disso, as informações relevantes do código-fonte da DSL são transformadas e escritas nesse arquivo.

A Listagem 2 ilustra uma representação de codificação em alto nível da função de processamento da GMaVis, onde é realizada a leitura de cada bloco do disco, filtragem, classificação e, finalmente a gravação dos dados no arquivo de saída. Para que isso ocorra, as operações iniciais são executadas entre as Linhas 5 e 6 e definem o inicio da região de leitura de dados. Enquanto que nas Linhas 7 e 8 os diferentes blocos de arquivos são processados, gerando uma *string* de dados com base na operações de filtragem e classificação. A última etapa consome a *string*, com base nas operações executadas nas Linha 9 realizando a gravação no arquivo de saída.

```
1 function process(args...){
2   Open(in_file);
3   Open(out_file);
4   while(!in_file.eof()){
5     d_size = Split(in_file);
6     d = Read(in_file,d_size);
7     f = Filter(d);
8     c = Classify(f);
9     Write(c,out_file);
10  }
11  Close(in_file);
12  Close(out_file);
13 }
```

Listagem 2. Representação da função de processamento em alto nível.

O compilador do GCC é chamado para criar o executável de pré-processamento de dados. Após a compilação, o compilador da DSL chama o módulo de pré-processamento de dados e espera que sejam realizadas as transformações de dados, apresentando os dados pré-processados. Esta saída é carregada no gerador de visualização que usa as informações armazenadas a partir do código fonte. O analisador da DSL fornece informações sobre os detalhes especificados no bloco de configuração, como tamanho, título, texto do marcador e visualizações a serem criadas.

4. Modelagem Paralela do Pré-Processamento na DSL GMaVis

Conforme descrito por [Ledur et al. 2017], a DSL GMaVis permite suporte à visualização geoespacial somente em arquiteturas *multi-core* através da geração de anotações de código [Griebler et al. 2017]. Desse modo, para habilitar o suporte ao processamento paralelo e distribuído para o módulo de pré-processamento de dados, foram implementadas três variantes do modelo Mestre/Escravo. Uma versão em MPI estática, uma versão com distribuição dinâmica de tarefas e outra usando o MPI-I/O. Essas versões foram denominadas respectivamente: MPI-E, MPI-D e MPI-I/O.

O MPI-E implementa a versão estática do modelo Mestre/Escravo em MPI. Nas Listagens 3 e 4 são apresentados os trechos de código das implementações Mestre e Escravo da versão MPI-E, respectivamente. Note que o processo principal (Mestre) realiza uma contagem inicial da quantidade de conjuntos de dados a serem processados. Após essa contagem, é realizada a divisão proporcional dos conjuntos de dados disponíveis entre os processos (Escravos). Cabe destacar que o processo Mestre também exerce a

função de Escravo, além da sua tarefa usual de coordenação e envio de trabalho aos processos Escravos. Outro aspecto importante diz respeito aos conjuntos de dados, que são distribuídos entre os processos de forma aleatória, não havendo nenhuma análise previa.

```
MPI_E_MASTER(args...){
    files<path>;
    numFiles = files.size()/np;
  for(k=1;k<np;k++){
      MPI::Send(numFiles);
      for(i=0;i<numFiles;i++){
          MPI::Send(files.pop());
      }
  }
  while(!files.empty()){
      file = files.pop();
      process(file);
    }
}
```

Listagem 3. MPI-E (Processo Mestre).

```
MPI_E_SLAVE(args...){
    files<path>;
    MPI::Recv(numFiles);
  for(i=0;i<numFiles;i++){
      MPI::Recv(path);
      files.add(path)
  }
  while(!files.empty()){
      file = files.pop();
      process(file);
    }
}
```

Listagem 4. MPI-E (Processo Escravo).

```
MPI_D_MASTER(args...){
    files<path>;
    numFiles = files.size()/np;
  for(k=1;k<np;k++){
      MPI::Send(files.pop());
  }
  for(i=0;i<numFiles;i++){
      MPI::Recv(free_slave);
      if(!files.empty()){
          MPI::Send(files.pop());
      }else{
          MPI::Send(end_message);
      }
    }
}
```

Listagem 5. MPI-D (Processo Mestre).

```
MPI_D_SLAVE(args...){
    MPI::Recv(file,tag);
    while(tag != end_message){
        function_process(file);
        MPI::Send(free_slave);
        MPI::Recv(file,tag);
    }
}
```

Listagem 6. MPI-D (Processo Escravo).

A versão dinâmica do módulo de pré-processamento de dados implementa o modelo Mestre/Escravo denominado MPI-D. A versão dinâmica se refere a habilidade do processo Escravo solicitar tarefas dinamicamente. Isso não deve ser confundido com a capacidade de criação de novos processos (dinamicamente) em tempo de execução, característica obtida através do uso da função *MPI_Comm_spawn* que não foi utilizada, presente desde a versão 2 do MPI. Assim, o processo principal (Mestre) do MPI-D realiza a distribuição dos conjuntos de dados entre os processos Escravos, conforme demonstrado na Listagem 5. Um vez realizada a fase de distribuição, inicia-se a fase de espera por parte do Mestre, que fica no aguardo do término da tarefa por parte de algum Escravo (demonstrado na Listagem 6), para somente então enviar uma nova tarefa para o Escravo. O aspecto dinâmico, está centrado no comportamento do Escravo, ou seja, a iniciativa

de solicitar uma nova tarefa parte dinamicamente do Escravo. Obviamente, isso ocorre somente se ele já finalizou sua tarefa anteriormente recebida.

Finalmente, a versão utilizando MPI-I/O é descrita. É importante ressaltar que a versão de MPI-I/O difere em alguns aspectos das demais versões (MPI-E e MPI-D), pelo fato de ter como principal forma de comunicação a troca de endereços entre os processos, pois há apenas um *único arquivo* compartilhado, com cada Escravo lendo e escrevendo em uma parte do arquivo. Assim, o processo principal (Listagem 7) calcula o tamanho total do conjunto de dados, realiza a divisão em partes (através da função *calculateJob()*) e envia o primeiro endereço do arquivo a ser processado para o primeiro escravo, representado na Listagem 8, e assim divide o arquivo com os demais processos, podendo ter novas iterações caso o número de *chunks* seja superior ao número de processos.

```
MPI_IO_MASTER(args ...){
  MPI::Send(primeiroPonteiro);
  while(hasJob){
    MPI::Recv(tag);
    if(tag==ponteiro){
        calculateJob();
        if(hasJob){
            getFreeSlave();
            MPI::Send(ponteiro);
        }else{
            MPI::Send(endSignal);
         }
     }else if(tag==endRead){
        if(hasJob){
            MPI::Send(ponteiro);
        }else{
            MPI::Send(endSignal);
         }
    }
  }
}
```

Listagem 7. MPI-IO (Mestre).

```
MPI_IO_SLAVE(args ...){
  MPI::File in;
  do{
      MPI::Recv(tag);
      if(tag==ponteiro){
        inicio=ponteiro;
        fim = ponteiro + chunk;
        buffer =
              in.Read(inicio,fim);
        ptrRetorno =
            ajustaRegistro(buffer);
        MPI::Send(ptrRetorno);
        process(buffer);
      }else if(tag==endSignal){
        MPI::Finalize();
        }
    } while (tag != endJob);
}
```

Listagem 8. MPI-IO (Escravo).

Nesse caso, cada processo escravo realiza a leitura, processamento e escrita na parte designada a cada um. O Mestre continua sendo responsável pela distribuição de tarefas. No entanto, durante interações envia endereços a serem processados para eventuais processos escravos disponíveis (através da função *getFreeSlave()*). Feito isso, realizam o processamento, classificação e filtragem dos dados. Em outras palavras, a implementação busca dividir o arquivo em *chunks* para cada escravo processar uma parte de forma independente e melhorar o desempenho a partir da leitura e escrita em partes. Os diversos arquivos de entrada são agrupados em um único arquivo compartilhado. Cada escravo lê e escreve no arquivo compartilhado através de troca de mensagens.

5. Experimentos

Para a realização dos experimentos, foram utilizados essencialmente três conjuntos de dados obtidos através do *Yahoo Flickr Creative Commons* [Thomee et al. 2016]: um conjunto de dados denominado de grande (4GB), um médio (1GB) e um pequeno (200MB).

Além disso, os experimentos executados para verificar o desempenho utilizaram um número diferente de réplicas (grau de paralelismo) dependendo do tipo de carga (balanceada ou desbalanceada), sendo realizadas 15 repetições para cada experimento, as diferentes em nenhum cenário apresentaram um desvio padrão superior a 2%. O tamanho de cada *chunk* usado em todos os experimentos foi definido como 100 MB, sendo um tamanho de *chunk* representativo para os cenários testados.

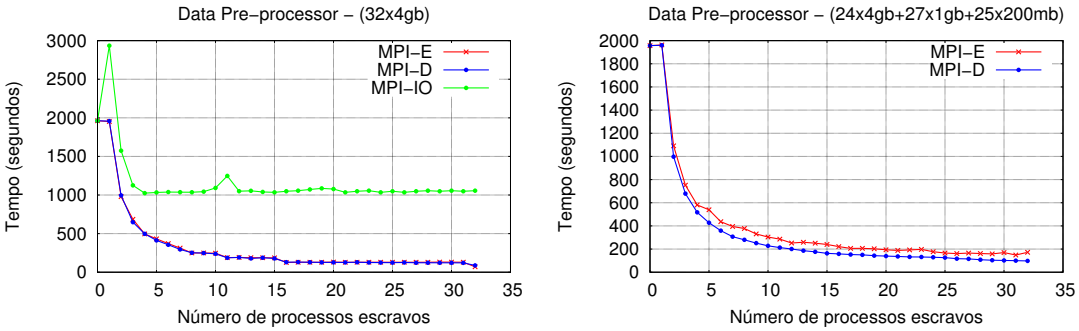
Os testes com carga balanceada utilizaram 32 réplicas de cada arquivo com carga grande (32x4GB), carga média (32x1GB) e carga pequena (32x200MB) para MPI-E, MPI-D e MPI-I/O. Por sua vez, os testes com carga desbalanceada combinaram o uso de diferentes cargas para MPI-E, MPI-D e MPI-I/O. Onde a carga grande (24x4GB + 27x1GB + 25x200MB), a carga média (6x4GB + 6x1GB + 10x200MB) e a carga pequena (1x4GB + 2x1GB + 2x200MB) foram definidas intencionalmente de modo a simular desbalanceamento dos conjuntos de dados. O número de arquivos bem como seus tamanhos foram definidos para que o resultado da soma dos arquivos totalizasse a mesma quantidade de dados (em Gigabytes) da carga balanceada.

Em relação ao ambiente de execução, a infraestrutura consistiu de quatro nodos, sendo que cada nodo possui uma configuração com dois processadores Intel Xeon E5520 - 2.27GHz (cada um com 4 cores e 8 *Hyper-Threads*) e 16GB de memória RAM. A rede de interconexão utilizada foi uma Gigabit Ethernet. Além disso, utilizou-se o sistema operacional Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-86-genérico x86 64) e o módulo de pré-processamento de dados foi compilado a partir do GCC-5.3.0 sem nenhuma *flag* de otimização definida.

É importante ressaltar que não foram realizados testes com carga desbalanceada para as versões de MPI-I/O, devido a implementação agrupar diferentes arquivos de entrada em um único arquivo e distribuir o processamento enviando *chunks* para os processos escravos, cada processo lê e escreve em uma porção do arquivo. Sendo assim, a versão de MPI I/O é efetiva também no cenário com diferentes tamanhos de arquivos. No entanto, o agrupamento e divisão desses arquivos em um único evita o desbalanceamento de carga, o desempenho seria o mesmo com arquivos de tamanhos idênticos.

A Figura 2 ilustra as versões paralelas do módulo de pré-processamento de dados executando com carga de trabalho grande (balanceada e desbalanceada). A execução com 0 processos paralelos representa a execução sequencial. Na Figura 2(a) é possível observar o resultado do experimento com o uso da carga de trabalho balanceada. Note que MPI-E e MPI-D apresentaram uma curva de desempenho semelhante. A exceção ficou por conta de MPI-I/O que apresentou um desempenho inferior. No outro cenário apresentado na Figura 2(b), com carga de trabalho desbalanceada, MPI-D se mostrou mais eficiente que MPI-E. Em relação a MPI-E, a diferença (pequena) de desempenho se torna mais evidente a partir do uso de 5 processos escravos.

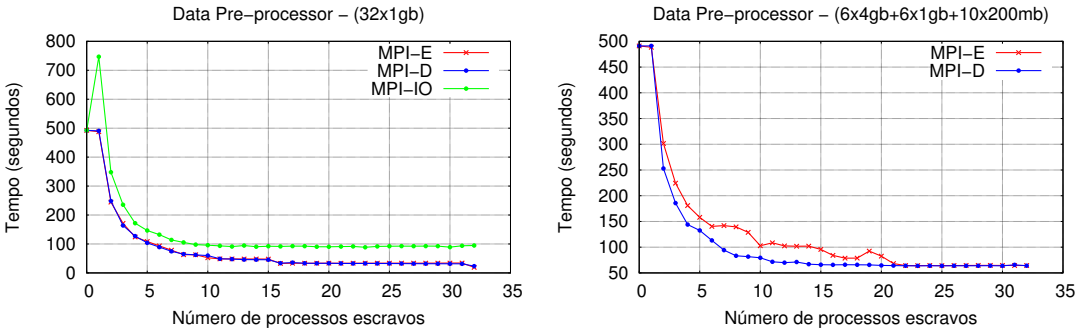
As versões paralelas do módulo de pré-processamento de dados executando com carga de trabalho média (balanceada e desbalanceada) são apresentadas na Figura 3. O experimento com carga balanceada (Figura 3(a)) apresentou novamente uma curva de desempenho praticamente igual para MPI-E e MPI-D. MPI-I/O se mostrou mais uma vez menos eficiente. No entanto, é possível perceber uma aproximação de MPI-I/O com base em sua curva de desempenho. O experimento desbalanceado (3(b)) por sua vez, sinaliza



(a) Carga grande balanceada. (b) Carga grande desbalanceada.

Figura 2. Módulo de pré-processamento de dados paralelo com carga grande.

uma tendência, onde as curvas de desempenho de MPI-E, MPI-D e MPI-I/O praticamente convergiram. MPI-D continuou como a versão mais eficiente, conseguindo superar em eficiência MPI-E entre 3 e 21 processos escravos. O tempo total para execução de MPI-I/O continuou sendo maior.



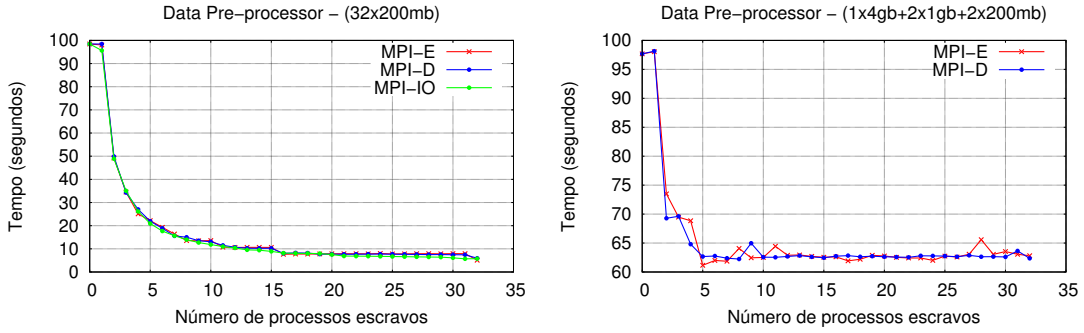
(a) Carga média balanceada. (b) Carga média desbalanceada.

Figura 3. Módulo de pré-processamento de dados paralelo com carga média.

Para o último cenário, mostrado na Figura 4, foram executadas versões paralelas do módulo de pré-processamento de dados com carga de trabalho pequena (balanceada e desbalanceada). A Figura 4(a) apresenta o experimento realizado com cargas balanceadas. Nesse cenário, pode-se dizer que as três versões (MPI-E, MPI-D e MPI-I/O) tiveram resultados semelhantes de desempenho. Porém, o experimento realizado com carga pequena balanceada mostra o cenário MPI-I/O com desempenho levemente superior aos outros cenários. A sua curva de desempenho foi mais eficiente em relação a utilização de recursos de MPI-E e MPI-D. Esse comportamento se torna mais evidente a partir do uso de 5 processos escravos, se mantendo assim até o final da execução.

A partir dos resultados obtidos, é possível identificar tendências e perfazer algumas considerações. A versão de MPI-D claramente se mostrou mais eficiente com o uso de cargas grandes e médias (sejam balanceadas ou desbalanceadas). Esse resultado de certa forma já era esperado, uma vez que o comportamento dinâmico na distribuição das tarefas de MPI-D possibilita uma melhor utilização dos recursos. No entanto, com o uso de cargas pequenas balanceadas o MPI-I/O apresentou crescimento de desempenho da versão de MPI-I/O. A curva de desempenho mostrou uma utilização de recursos bastante

acentuada. Isso ocorre devido ao MPI-I/O operar sobre um único arquivo compartilhado com todos os processos, que com tamanho pequeno tiveram um melhor balanceamento. Por outro lado, o MPI-I/O com um arquivo maior, tem desempenho inferior, limitado pelo *chunk* do arquivo mais lento processado nos escravos e também pelo gargalo de desempenho nas operações de *read* e *write* de disco no arquivo compartilhado.



(a) Carga pequena balanceada. (b) Carga pequena desbalanceada.

Figura 4. Módulo de pré-processamento de dados paralelo com carga pequena.

6. Conclusões

Neste artigo foi apresentada uma proposta de suporte ao processamento paralelo e distribuído, com foco em uma DSL para visualização de grandes conjuntos de dados geoespaciais. Para isso, foram avaliadas diferentes implementações do padrão de programação distribuído MPI. O pré-processamento de dados é a etapa mais demorada para visualização de dados. Assim, o uso do processamento distribuído torna possível evitar gargalos de desempenho de disco em um único nodo e potencialmente aumentar o desempenho com diversas operações simultâneas em diferentes nodos.

Os resultados obtidos demonstram um desempenho superior da versão MPI-D com o uso de cargas de tamanho grande e médio. A abordagem de distribuição dinâmica de tarefas permitiu uma melhor utilização dos recursos, com cargas balanceadas e desbalanceadas. Por outro lado, a versão de MPI-I/O obteve o melhor resultado com cargas pequenas, mostrando que a carga de trabalho não foi limitada pelo subsistema de E/S, não havendo gargalos de desempenho devido ao seu tamanho reduzido.

Futuramente, um objetivo é implementar novas técnicas em implementações do MPI-I/O. Ainda, pretende-se implementar uma versão de MPI adaptativo, ou seja, a versão deverá permitir a criação de processos em tempo de execução, permitindo assim explorar a elasticidade horizontal do ambiente. Para isso, pretende-se fazer uso do estilo de programação dinâmica do MPI-2.

Agradecimentos

Os autores gostariam de agradecer o suporte financeiro parcial das pesquisas para as seguintes instituições: PUCRS, CAPES e CNPq.

Referências

[Ayachit 2015] Ayachit, U. (2015). *The ParaView Guide: A Parallel Visualization Application*. Kitware, Inc., USA.

[Griebler et al. 2017] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):20.

[IDC 2012] IDC (2012). The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East.

[Kraak and Ormeling 2011] Kraak, M. and Ormeling, F. (2011). *Cartography, Third Edition: Visualization of Spatial Data*. Guilford Publications.

[Latham et al. 2017] Latham, R., Bautista-Gomez, L., and Balaji, P. (2017). Portable Topology-Aware MPI-I/O. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 710–719.

[Ledur et al. 2017] Ledur, C., Griebler, D., Manssour, I., and Fernandes, L. G. (2017). A High-Level DSL for Geospatial Visualizations with Multi-core Parallelism Support. In *41th IEEE Computer Society Signature Conference on Computers, Software and Applications*, COMPSAC'17, Torino, Italy. IEEE.

[Ledur et al. 2015] Ledur, C., Griebler, D., Manssuor, I., and Fernandes, L. G. (2015). Towards a Domain-Specific Language for Geospatial Data Visualization Maps with Big Data Sets. In *ACS/IEEE International Conference on Computer Systems and Applications*, AICCSA'15, page 8, Marrakech, Marrocos. IEEE.

[Mendez et al. 2017] Mendez, S., Rexachs, D., and Luque, E. (2017). Analyzing the Parallel I/O Severity of MPI Applications. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 953–962.

[Moreland 2013] Moreland, K. (2013). A Survey of Visualization Pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):367–378.

[Perrot et al. 2015] Perrot, A., Bourqui, R., Hanusse, N., Lalanne, F., and Auber, D. (2015). Large Interactive Visualization of Density Functions on Big Data Infrastructure. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 99–106.

[Rünger and Rauber 2013] Rünger, G. and Rauber, T. (2013). *Parallel Programming - for Multicore and Cluster Systems; 2nd Edition.* Springer.

[Seo et al. 2015] Seo, S., Latham, R., Zhang, J., and Balaji, P. (2015). Implementation and Evaluation of MPI Nonblocking Collective I/O. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 1084–1091.

[Steed et al. 2013] Steed, C. A., Ricciuto, D. M., Shipman, G., Smith, B., Thornton, P. E., Wang, D., Shi, X., and Williams, D. N. (2013). Big Data Visual Analytics for Exploratory Earth System Simulation Analysis. *Comput. Geosci.*, 61:71–82.

[Thomee et al. 2016] Thomee, B., Shamma, D. A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D., and Li, L.-J. (2016). YFCC100M: The New Data in Multimedia Research. *Commun. ACM*, 59:64–73.

[Wylie and Baumes 2009] Wylie, B. N. and Baumes, J. (2009). A Unified Toolkit for Information and Scientific Visualization. In *VDA*, page 72430.

[Zhang et al. 2015] Zhang, T., Hua, G., and Ligmann-Zielinska, A. (2015). Visually-driven Parallel Solving of Multi-objective Land-use Allocation Problems: A Case Study in Chelan, Washington. *Earth Science Informatics*, 8:809–825.