Fundação Universidade Federal do Rio Grande
Modelagem Computacional

MPnetCDF: Uma Proposta de Paralelização do NetCDF

Jeison Luiz Ferreira Vieira

Rio Grande, 26 de Outubro de 2009

Fundação Universidade Federal do Rio Grande
Modelagem Computacional

MPnetCDF: Uma Proposta de Paralelização do NetCDF

Jeison Luiz Ferreira Vieira

Trabalho de Conclusão do Curso de Mestrado em Modelagem Computacional submetido à avaliação, como requisito parcial à obtenção do título de Mestre em Modelagem Computacional.

Orientadora: Profª. Dra. Silvia Silva da Costa Botelho

Co-orientador: Prof. Dr. Nelson Lopes Duarte Filho

Rio Grande, 26 de Outubro de 2009

Este trabalho foi analisado e julgado adequado para a obtenção do título de Mestre em Modelagem Computacional e aprovado em sua forma final pelo orientador.

_____

Profª. Dra. Silvia Silva da Costa Botelho

Banca Examinadora:

Profª. Dra. Sílvia Silva da Costa Botelho

C3 – FURG (Orientadora)

Prof. Dr. Nelson Lopes Duarte Filho

C3 – FURG (Co-orientador)

Profª. Dra. Elisa Helena Leão Fernandes

Oceanografia Física – FURG

Prof. Dr. Adenauer Corrêa Yamin

Ciência da Computação –UCPel

Dedico este trabalho ao meu caro amigo José de Castro, que me ensinou vivamente que todo conhecimento especializado só tem valor real quando relacionado ao todo, cotextualizado na vida e posto a serviço do próximo.

Agradecimentos

Agradeço à Deus.

Agradeço aos meus orientadores.

Agradeço aos amigos William e Andrea Berneira, que se encarregaram da tradução do resumo.

Agradeço a todos que, de alguma forma, me ajudaram para que esse trabalho fosse concluído.

Conteúdo

Lista de Figuras

Lista de Abreviaturas

BDB Berkeley Database

DRDB Disk Resident Database System

MMDB Main Memory Database System

MPI Message Passing Interface

RPC Remote Procedure Call

HD Hard Disk

BD Banco de Dados

E/S Entrada e Saída

SGBDM Sistema Gerenciador de Banco de Dados em Memória

Resumo

Atualmente, encontram-se disponíveis arquiteturas computacionais paralelas e de baixo custo. Qualquer conjunto de computadores interligados por uma rede pode funcionar em conjunto visando a resolução de uma tarefa, constituindo assim, um *cluster* de computadores. Entretanto, essas arquiteturas, que oferecem, como recursos, maior capacidade de processamento e armazenamento, não são inteiramente aproveitadas por dois motivos: I) a cultura de centralização por parte dos usuários, habituados com a execução de suas tarefas em uma única estação de trabalho, e II) pela falta da existência de ferramentas computacionais que ofereçam a possibilidade de distribuição das tarefas entre computadores.

Essa limitação foi herdada por diversas ferramentas, como o Network Commom Data Form, NetCDF, que é uma especificação, desenvolvida pela fundação Unidata e oferecida sob a forma de bibliotecas utilitárias, visando armazenar grandes volumes de dados gerados principalmente na área da geociência. Ela especifica como armazenar os dados na forma de um conjunto de matrizes inter-relacionáveis e auto-descritivas. Ou seja, dados auxiliares que expressam como os dados em si devem ser interpretados, também ficam armazenados no arquivo netCDF, chamado dataset.

Buscando ampliar as capacidades desse sistema, surgiram algumas iniciativas para o desenvolvimento de versões que facilitam o uso paralelo dos datasets, tais como o Parallel NetCDF e o NetCDF-4/HDF5, ambas desenvolvidas utilizando-se MPI-IO como interface de E/S paralela, porém com diferentes abordagens quanto a divisão e gerenciamento dos dados.

Neste trabalho é proposta a implementação de um sistema netCDF paralelo, desenvolvido a partir de sua versão clássica, utilizando, de forma original, o gerenciador de banco de dados em memória Berkeley DataBase (BDB). Tal proposta, denominada

MPnetCDF em virtude de ser uma implementação paralela em memória, foi desenvolvida, testada e validada em diferentes cenários, apresentando um conjunto de vantagens, tais como maior rapidez na realização de operações de escrita e leitura em ambientes distribuídos, compatibilidade com aplicações desenvolvidas para netCDF clássico e maior flexibilidade para representar e descrever situações físicas reais.

Abstract

At present, there are available low cost parallel computational networks. Any group of computers connected in a network can work in unison to complete a task. This network would be considered a *cluster* computer. Although, these types of network offer larger processing capabilities and data storage, they are not commonly used for two reasons: I) the culture of centralized work where all data and tasks are completed in one workstation; II) the lack of computational tools to share the tasks and the data among the network.

These inherent limitations are present in diverse tools such as Network Common Data Form (NetCDF). This format, developed by Unidata Foundation, offers access libraries with the objective of store large volumes of data generated mainly in the area of Geoscience. The NetCDF specifies how to store data in a group of interrelated and auto-descriptive matrices. This means that the accessory data which express how to interpret the data is also stored in the file NetCDF and it is called dataset.

With views to increase the capacity of this system, versions were developed to facilitate the parallel use of datasets such as Parallel NetCDF and NetCDF-4/HDF5. Both were developed using MPI-IO as an interface of Parallel E/S. However there are different approaches in the division and management of data.

This thesis proposes the implementation of a Parallel NetCDF, developed from its classic version, using the original format of memory data bank management, Berkeley DataBase (BDB). The proposed network because of its parallel implementation of memory is known as MPnetCDF. This was developed, tested and validated in different scenarios, presenting advantages such as high speed reading and writing operations, compatibility with applications developed for the classic NetCDF and improved flexibility to represent and describe real physical situations.

Capítulo 1

Introdução

Aliando-se a tendência de aumento da capacidade das memórias dos computadores e da diminuição dos seus custos à possibilidade de interligar computadores através de redes com altas taxas de transmissão de dados, também a baixo custo, obtem-se um cenário favorável às aplicações distribuidas, especialmente àquelas críticas em relação à performance, que poderão ter todos ou grande parte dos seus dados armazenados nas memórias principais dos nodos de um agregado (*cluster*) de computadores. Isso traz vantagens óbvias de desempenho, em função dos baixos tempos de acesso a memória volátil, quando comparada aos meios de armazenamento magnéticos, como discos rígidos.

Entretanto, essas arquiteturas atualmente ainda costumam ficar subutilizadas. Tanto devido à falta de cultura para elaboração de sistemas paralelos e distribuídos, quanto devido a ferramentas hoje disponíveis para tal, que são de uso inamistoso, a exemplo do MPI, que é um padrão para comunicação de dados em computação paralela, oferecido sob a forma de uma biblioteca de funções disponível para diversas linguagens de programação.

Neste panorama, surgem novas propostas que buscam explorar o poencial dos sistemas distribuídos. [Filho and Pedone, 2008], por exemplo, apresenta uma abordagem baseada em DSM (Distributed Shared Memory) [Nitzberg and Lo, 1991] como infraestrutura para compartilhamento e distribuição dos dados. Em especial, citamos sistemas de gerenciamento de espaços de armazenamento distribuídos. Tais sistemas visam oferecer um espaço único de endereços às aplicações, apesar desse espaço realizar-se sobre as memórias dos vários computadores que constituem o cluster. Propostas como estas costu-

mam perseguir soluções genéricas, independentes das aplicações e, portanto, apresentam vantagens como serem transparentes e de mais fácil utilização, por garantirem aos programadores uma visão próxima a dos sistemas centralizados. Por outro lado, apresentam custos como, por exemplo, o necessário a garantir modelos de consistência que admitam aplicações genéricas concorrentes.

Neste trabalho, se propõe uma abordagem intermediária a essa, que objetiva manter a visão de espaço de armazenamento compartilhado, mas um tanto voltada para as características da aplicação, com um grau de generalidade menos abrangente, mas de menor esforço de implementação.

O sistema distribuído proposto, denominado BDBNetCDF, se constitui de: I- uma camada de serviço, implementada a partir do Berkeley DataBase - BDB [Sleepycat-Group, 2009], banco de dados capaz de gerenciar tabelas totalmente em memória, oferecendo acesso remoto através da tecnologia cliente-servidor; II- uma camada de distribuição que oferece uma API voltada para as características da aplicação; e III- a camada de aplicação propriamente dita. A camada de distribuição oferece um modelo de consistência adequado às necessidades da aplicação, com isso restringindo a generalidade do gerenciador de armazenamento, em favor da possibilidade de se utilizar um sistema de banco de alta performance, o BDB.

A idéia de se utilizar essa arquitetura surgiu da percepção de que uma especificação altamente utilizada nas geociências, o Network Commom Data Form - NetCDF [Rew et al., 2008b], padrão de armazenamento de imensos volumes de dados, originalmente implementado de forma centralizada, pode ser tornado distribuído, sem que os seus usuários necessitem mudar o paradigma de programação até então utilizado, permitindo que mesmo programas desenvolvidos para o padrão clássico centralizado possam tirar proveito da distribuição dos dados aqui descrita, de modo transparente.

No capítulo seguinte é descrito o NetCDF e, no capítulo 3, o BDB, ambos de forma simplificada, com vistas a apenas introduzir os conceitos básicos necessários a melhor compreensão da proposta apresentada. No capítulo 4 é descrito o BDBNetCDF, conforme ele foi implementado, e no capítulo 5 apresentam-se alguns resultados com ele obtidos. Finalmente, o capítulo 6 conclui o trabalho, descrevendo algumas sugestões de trabalhos futuros.

Capítulo 2

O netCDF

Este capítulo tem objetivo descrever o sistema que propõe-se distribuir, o netCDF. Inicialmente será feita uma apresentação geral do sistema e em seguida serão tratadas as difenças entre as principais versões. Nas seções 2.3 e 2.4 serão descritos a forma de representação dos dados do sistema e a estrutura de armazenamento dos arquivos, respectivemente. Por fim, é abordada a questão de acesso concorrente aos dados por diversos processos.

2.1 Apresentação

O netCDF é uma especificação de software implementada pela Unidata [Unidata, 2009] sob a forma de uma biblioteca de funções de acesso a dados, armazenados na forma de matrizes [1]. Uma matriz é uma estrutura retangular n-dimensional (onde n é 0, 1, 2, ...) contendo itens os quais tem todos o mesmo tipo (ex.: caracteres de 8 bits ou de 32 bits). Um escalar é uma matriz 0-dimensional. Um arquivo contendo o conjunto de dados referente a uma aplicação é denominado **dataset**.

O netCDF oferece uma visão dos dados como uma coleção de objetos auto-descritivos e portáteis que podem ser acessados através de uma interface simples. "Auto-descritivos" significa que o dataset pode conter informações que definem os dados que ele mantém, como as unidades de medida a eles associadas. "Portáteis" significa que os dados em um dataset são representados em uma forma que pode ser acessada por sistemas

[1] O termo **matriz** será empregado para referenciar estruturas multidimensionais em geral

computacionais de diferentes arquiteturas. Os valores nas matrizes podem ser acessados diretamente, sem a necessidade de conhecimento a respeito de como eles são armazenados. Aplicações podem acessar os dados de forma a transformar, combinar, analisar ou mostrar campos específicos.

O netCDF implementa um tipo abstrato de dados, o que significa que todas as operações de acesso e manipulação de dados precisam ser feitas através do conjunto de funções oferecidas pela interface. A forma de representação dos dados é transparente para a aplicação que usa a interface. Assim, a maneira como os dados são armazenados pode ser alterada sem que se necessite alterar os programas já existentes. Essa propriedade foi explorada na realização deste trabalho, permitindo que as funções primitivas de manipulaçao de arquivos do sistema operacional originalmente utilzadas pelo netCDF fossem substituídas pelas funções oferecidas pelo Berkeley DataBase, sem modificação da interface existente.

Por exemplo, a função para criação de uma dataset, nc_create, tem o seguinte formato:

int nc_ create(char *path, int cmode, int *ncidp)

Ela retorna um inteiro e recebe os parâmetros: nome do arquivo que conterá o dataset, tipo de dataset e o endereço de memória onde ficará armazenada a idendificação do dataset, respectivamente. A versão modificada aqui apresentada mantém o mesmo formato e a criação de um dataset é expressa no código, da mesma maneira que na versão original.

A Unidata mantém interfaces do netCDF em C, Fortran 77, Fortran 90, C++ e, mais recentemente, em Java. Bibliotecas estão disponíveis para vários sistemas operacionais baseados em UNIX e também para Windows.

2.2 Os Formatos de Arquivos NetCDF

Até a versão 3.6.0, o netCDF usava apenas um formato binário para representar os dados, que será referido como *formato clássico*. Este é o formato padrão para todas as

versões do netCDF.

Na versão 3.6.0 um novo formato de 64 bits foi introduzido. Ele é praticamente idêntico ao formato clássico, com a diferença de poder tratar arquivos maiores, endereçados por 64 bits. Esses dois modelos são frequentemente referidos na documentação oficial como **NetCDF-3**.

Na versão 4.0.0 um novo formato binário surgiu, o formato HDF5 (ver seção 2.5), que trouxe consigo a eliminação de algumas restrições existentes nos formatos anteriores, como datasets de tamanho ilimitado e possibilidade de criação de mais de uma dimensão ilimitada por dataset (ver seção 2.3.1). Esse modelo é referido na documentação oficial como **NetCDF-4**.

Para que os dados possam ser válidos, independente da arquitetura do sistema computacional, os formatos clássico e 64 bits são implementados através de uma representação externa muito parecida com XDR (*eXternal Data Representation*, [Sirinavasan, 1995]), um padrão para descrição de dados.

2.3 O Modelo Descritor de Dados do NetCDF

Os dados são armazenados no netCDF em um **dataset**, um arquivo que contém três informações principais para descrever os dados: **dimensões**, **variáveis** e **atributos**, todas possuindo um nome e um número de identificação pelos quais podem ser referenciadas. Esses componentes são usados conjuntamente para dar significado aos dados e às relações entre os campos das matrizes existentes nos datasets.

As dimensões são as componentes de uma variável e são, geralmente, usadas para representar dimensões físicas reais, como tempo, latitude, longitude ou altura. A unidade básica referenciável, por nome, em um dataset é uma variável. Quando uma variável é definida, a sua forma é especificada como uma lista de dimensões. Por exemplo, a variável "ponto_geográfico" poderia ser descrita pelas dimensões "altitude", "latitude" e "longitude". Os atributos são usados para armazenar dados sobre os dados, os chamados metadados e, via de regra, ficam vinculados às variaveis que contém os dados aos quais referem-se. No exemplo citado, a variável poderia ter um atributo "unidades" associado, que contivesse uma informação como: "altitude = metros acima do nível do mar, latitude

= graus positivos ao norte e longitude = graus positivos a oeste".

Ao acessar um dataset, existem dois possíveis modos de operação: o modo de definição e o modo de dados. No primeiro, podem-se criar as dimensões, variáveis e novos atributos, mas não é permitido fazer escrita ou leitura de dados. No segundo, é possível acessar os dados ou mudar atributos existentes, mas as operações do modo anterior ficam indisponíveis.

2.3.1 Dimensões

Uma dimensão é composta por um nome e um comprimento. O comprimento é um número inteiro positivo, a exceção das dimensões que forem do tipo *UNLIMITED*, ilimitada. Uma dimensão é chamada ilimitada ou *record dimension* (dimensão de registros) quando os dados por ela indexados forem coletados dinamicamente durante a execução do modelo, sem um valor limite pré-definido para indexação. Um exemplo de dimensão ilimitada, bastante utilizado em descrições físicas, é o tempo, em uma frequência de amostragem.

Um **registro** é o conjunto de dados obtidos para cada variação unitária do índice da dimensão ilimitada. Por exemplo, imaginemos uma matriz bidimensional na qual, a cada cinco minutos, sejam medidas as temperaturas em dez pontos de uma superfície. Sendo o tempo a dimensão ilimitada, teremos, para cada medição, dez temperaturas associadas ou, em outras palavras, um único registro composto de dez elementos. Ao final de uma hora, teríamos doze registros.

As variáveis são definidas, cada uma, por uma lista de dimensões, essas dimensões precisam já existir no momento da criação da variável. O número de dimensões de uma variável é chamado *rank*. Uma variável escalar tem rank 0, um vetor tem rank 1, uma matriz bidimensional tem rank 2 e, assim, sucessivamente.

Uma variável com uma dimensão ilimitada pode crescer livremente ao longo daquela dimensão. Os datasets no modelo clássico e 64 bits podem ter apenas uma dimensão ilimitada por dataset e, no máximo, uma por variável dentro do dataset. Se ela existir, deverá ser a primeira a ser declarada na lista de dimensões que definem a variável e ser aquela que apresenta a variação mais lenta em relação as demais dimensões da lista. Na seção 2.3.4 apresenta-se um exemplo para melhor entendimento das relações entre

dimensões, variáveis e atributos.

Em um dataset netCDF-4/HDF5, qualquer número de dimensões ilimitadas pode ser usada e não há restrições na ordem da lista de dimensões da variável. No modelo desenvolvido nesse trabalho não há limites para o número de dimensões ilimitadas declarados por dataset, contudo continuará podendo haver no máximo uma por variável e ela ainda deverá ser a que apresenta variação mais lenta.

2.3.2 Variáveis

As variáveis são usadas para armazenar os dados em si. Uma variável representa uma matriz de valores do mesmo tipo. Um escalar é tratado como uma matriz 0-dimensional. Cada variável é descrita por um nome, um tipo e uma forma associada à lista de dimensões, informados no momento da sua criação. Uma variável também pode ter atributos, os quais podem ser adicionados, apagados ou alterados depois da criação da variável.

Nos arquivos em formato clássico e 64 bits apenas seis tipos primitivos de dados estão disponíveis: *byte*, *character*, *short*, *int*, *float* e *double*. Variáveis em arquivos netCDF-4 também podem ser *unsigned short*, *unsigned int*, *64-bit int*, *unsigned 64-bit int*, *string* ou um tipo composto semelhante a uma struct da linhuagem C.

As variáveis que possuem a dimensão ilimitada são chamadas *record variables* (variável de registros). Uma variável que não seja do tipo record variable tem um tamanho fixo dado pelo produto dos tamanhos das suas dimensões. O tamanho de uma variável de registros também é obtido do mesmo modo, mas nesse caso o produto é variável porque envolve a dimensão ilimitada, a qual pode variar. O comprimeto da dimensão ilimitada é igual ao número de registros.

Ainda existe um tipo especial de variáveis, as de coordenadas. Variáveis desse tipo estão em função apenas de uma dimensão (rank = 1) e possuem o mesmo nome de uma dimensão. Elas definem uma coordenada física correspondente àquela dimensão, ou seja, uma posição ao longo de uma dimensão pode ser especificada por um índice, mas se a dimensão tiver uma variável de coordenada correspondente, então é possivel fazer o mapeamento, atravez de funções oferecidas na API do netCDF, entre o dado e o valor da coordenada.

2.3.3 Atributos

Os atributos são identificados pelo nome da variável juntamente com o nome do atributo. Alguns oferecem informação sobre o dataset como um todo e são chamados atributos globais, sendo identificados apenas pelos seus nomes, sem associação com uma variável.

Normalmente, um atributo está associado a uma variável (a excessão dos atributos globais) e possui um nome, um tipo, um tamanho e um valor. O tipo de um atributo é especificado quando ele é criado. Os tipos permitidos para os atributos são os mesmos permitidos para as variáveis.

Os atributos são mais dinâmicos do que as variáveis e dimensões; eles podem ser acrescidos, apagados e ter seus tipos, tamanhos e valores modificados depois de criados, enquanto que a interface netCDF não oferece possibilidade de apagar uma variável ou mudar seu tipo ou forma depois de criada.

Uma diferença que merece ressalva é que o total de metadados associados a um objeto netCDF, e armazenado em seus atributos, é normalmente pequeno e pode residir na memória volátil do sistema, enquanto que os dados atrelados as variáveis são, geralmente, muito grandes para caber na memória e ficam sujeitos aos mecanismos de paginação ([Tanenbaum, 2002]), o que traz custos relevantes em relação ao desempenho. No modelo aqui descrito, esse problema foi superado e tanto dados como metadados ficam residentes na memória, devido ao fato do Berkeley DB ser um gerenciador de banco de dados em memória (ver seção 3.1).

2.3.4 Exemplo

Usaremos um pequeno exemplo, retirado do manual do usuário do netCDF [Rew et al., 2008b] e reproduzido no quadro 2.1, para ilustrar os conceitos apresentados anteriormente sobre o modelo descritor de dados do netCDF.

A notação usada para descrever esse dataset é chamada *CDL* (network Common Data form Language). O pacote netCDF inclui um utilitário, o ncdump, para produção de texto em CDL a partir de datasets binários e vice versa.

O exemplo descreve a seguinte estrutura de dados:

- Quatro dimensões: lat, lon, level e time. As três primeiras são de tamanho fixo e time é a dimensão ilimitada;

- Seis variáveis; quatro são variáveis de coordenadas (apresentam rank = 1 e mesmo nome da dimensão): lat(lat), lon(lon), level(level) e time(time). As outras duas vairáveis (as vezes chamadas de variáveis primárias), temp (rank = 4) e rh (rank = 3), contêm os dados em si. Ambas são variáveis de registro, por possuirem uma dimensão ilimitada. Essa dimensão, conforme visto na seção 2.3.1, é a primeira declarada na lista de dimensões das respectivas variáveis e apresenta variação mais lenta em relação as demais, ou seja, tomando-se a variável rh como exemplo, para cada instante de tempo temos cinco variações de latitude e dez variações de longitude.

- Além do nome, cada uma das variáveis têm um tipo associado: *float* para temp e rh, *int* para lat, lon e level e *short* para time; elas também possuem uma forma dada pelas dimensões nomeadas entre parênteses ao lado de cada nome de variável.

- As quatro variáveis de coordenadas definem, seguindo a ordem em que aparecem: pressão barométrica, latitude, longitude e tempo. Assim, existem dados correspondentes à altitudes de 1000, 850, 700 e 500 milibars e em latitudes correspondentes a 20, 30, 40, 50 e 60 graus ao norte, da mesma forma para longitude e tempo. Desse modo, em vez de fornecer os índices para realizar uma consulta, por exemplo, a matriz rh, é possível fornecer, através de função oferecida pela API, os valores de latitude e longitude diretamente. Uma consulta à humidade relativa na latitude 30, longitude -15, resultaria em 0.8.

- Os atributos, por sua vez, aparecem declarados abaixo das variáveis. Vê-se, por exemplo, que, associado a variável latitude, existe, na linha quatorze, o atributo units que possue como valor o array de treze caracteres *degrees_ north* e valid_range é um atributo que aparece na linha doze do exemplo, associado a variável rh, composto por um vetor de dois elementos do tipo float, com os valores *0.0* e *0.1*. Na notação CDL os tipos dos atributos não são explicitamente declarados.

- Assim, com base nos metadados oferecidos pelos atributos e considerando a variável rh, que representa humidade relativa e é definida pela dimensões tempo, latitude

e longitude, vemos que decorridas *12* horas a partir da primeira hora do dia 01 de janeiro de 1996, temos dados relacionados na matriz bidimensional apresentada. Nessa hora, para uma latitude de *20* graus ao norte e uma longitude de *-15* graus a oeste, encontramos uma humidade relativa de *0.7*.

2.4 Estrutura do Arquivo

Não é necessário conhecer a estrutura do arquivo para realizar leituras e escritas de dados através do netCDF porque tais operações ficam escondidas sob a interface oferecida pelas funções. Entretanto, esse conhecimento, agora apresentado, ajudará a entender as limitações apresentadas pelo netCDF e as melhorias na proposta aqui descrita, conforme mostrado no capítulo 4.

Um dataset no formato clássico ou 64 bits é armazenado como um simples arquivo, dividido em duas partes: um **cabeçalho** e uma **parte de dados** (ver figura 2.1).

O cabeçalho, no início do arquivo, contém informações sobre as dimenções, variáveis e atributos no dataset, incluindo seus nomes, tipos e outras características. As informações a respeito de cada variável incluem o seu deslocamento em relação ao início da área de variáveis de tamanho fixo, ou o deslocamento relativo da variável na área de variáveis de registro. O cabeçalho também contém o tamanho das dimensões e informações necessárias ao mapeamento dos índices multidimensionais em deslocamentos apropriados dentro do arquivo.

Por padrão, o cabeçalho tem o espaço necessário para armazenar as informações sobre dimensões, variáveis e atributos do dataset e apenas um pequeno espaço extra, resultante do arredondamento do espaço para o bloco seguinte no disco. Isso traz como vantagem arquivos netCDF compactos, necessitando apenas de um *overhead* mínimo para armazenar os metadados que tornam os datasets auto-descritivos. Entretanto, uma desvantagem é que qualquer operação que resulte no aumento ou diminuição dos dados no cabeçalho, como por exemplo a adição de novas dimensões ou variáveis, requer que os dados sejam copiados para uma nova área de memória compatível com o novo tamanho. O mesmo problema ocorre ao se criar uma nova matriz de tamanho fixo. Como alternativa, é possivel especificar explicitamente, no momento da criação do dataset, um tamanho

extra a ser reservado para o caso de futuras mudanças.

Quando o tamanho do cabeçalho muda, dados no arquivo são movidos e o endereço dos dados também se altera. Se outro programa estiver lendo o netCDF durante essa redefinição, a sua visão será baseada nos antigos, e possivelmente incorretos, índices. Se houver a necessidade de uma redefinição enquanto o dataset estiver sendo compartilhado, então um mecanismo externo à biblioteca netCDF deverá ser empregado, para prevenir o acesso dos leitores durante a redefinição.

A parte de dados fica subdividida em duas. A primeira, corresponde aos dados de tamanho fixo, é utilizada pelas variáveis que não usam a dimensão ilimitada. Os dados relativos a cada variável são armazenados contiguamente nesta parte do arquivo. Se não houver variáveis de registro, esta será a última parte do arquivo netCDF.

A segunda parte segue a primeira e corresponde a um número variável de registros de tamanho fixo, cada um dos quais contém dados referentes às variáveis do tipo record. Os dados são armazenados contiguamente por registro e não por variável. Os registros das diversas variáveis ficam intercalados. Uma representação da estrutura do arquivo netCDF pode ser vista na figura 2.1.

Voltando ao exemplo apresentado no quadro 2.1, temos as variáveis lat, lon e level armazenadas na primeira parte de dados, após o cabeçalho, reservada para variáveis que não contenham a dimensão ilimitada. As variáveis temp, rh e time, por serem variáveis de registro, ficam na última parte do arquivo, de forma que o primeiro registro da variável temp, contendo os valores correspondentes de level, lat e lon, seja seguido pelo primeiro registro da variável rh, contendo os valores correspondentes de lat e lon, que será seguido pelo primeiro registro da variável de coordenada time, contendo um valor de tempo. O procedimento é repetido sucessivamente para os próximos registros de cada variável, até o tamanho máximo do dataset.

2.5 Acesso Concorrente aos Dados

Ao se permitir acessos simultâneos de vários processos ao mesmo conjunto de dados, deve-se ter o cuidado, especialmente para operações de escrita, de estabeler controle de forma que dois ou mais processos, operando sobre os mesmos conjuntos de dados, não

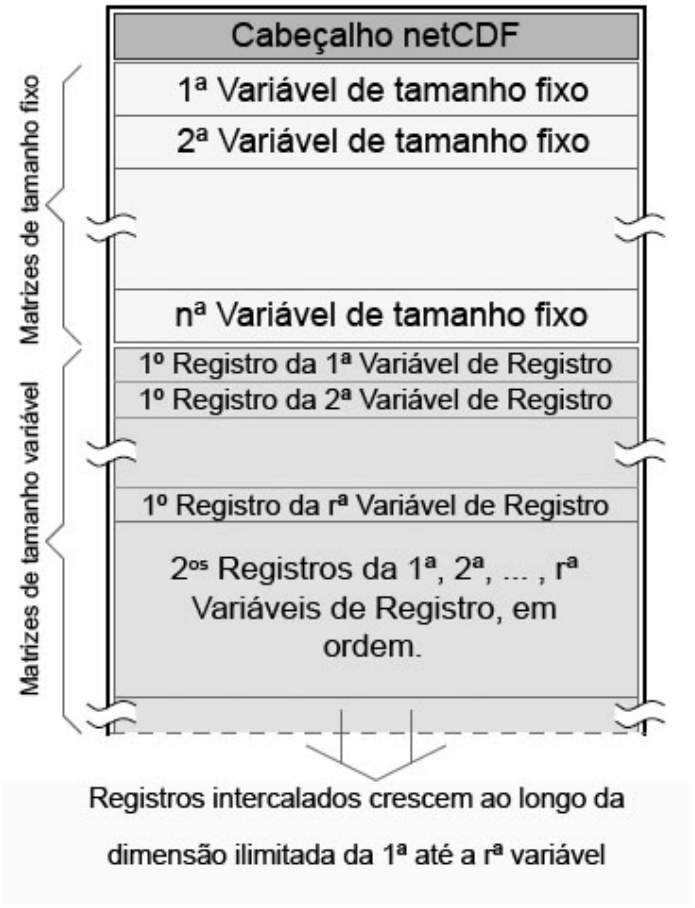

Figura 2.1: Representação da estrutura de um arquivo netCDF clássico.

venham a leva-los a um estado inconsistente. A figura 2.2, apresenta três cenários, cada um com quatro processos, de P0 até P3, tentando operar conjuntamente sobre os datasets instalados em um sistema de arquivos paralelo. Cada cenário, corresponde a uma possível solução para a questão, em relação ao netCDF.

Nas versões NetCDF-3 não há controle sobre a escrita simultânea de vários processos no mesmo arquivo. Se mais de um processo tentar tal procedimento, as requisições serão enfileiradas, conforme mostrado na figura 2.2(a). Para acessos simultâneos sobre o mesmo arquivo, a consistência é garantida apenas para múltiplos leitores ou um escritor. Por padrão, quando um dado é escrito, ele fica armazenado nos *buffers*[2] internos da apli-

[2]Buffers são regiões de memória temporária utilizadas para leitura e escrita de dados. Normalmente são utilizados quando há uma diferença entre a taxa em que são recebidos e a taxa em que são armazenados ou processados.

cação e só é gravado no disco rígido quando é feita uma chamada explicita a uma função responsável pela sincronização entre memória e HD.

Assim, para que não ocorram problemas de sincronização, ou seja, leituras no disco, de dados que foram modificados mas ainda não foram salvos, é possível abrir o dataset com *flags*[3] indicadoras de que haverá compartilhamento de informação, fazendo com que os dados não sejam temporariamente armazenados nos *buffers*. Mesmo assim, alguns cuidados por parte da aplicação ainda são necessários durante a execução de determinadas funções que exijam o redimensionamento do dataset e consequente modificação do cabeçalho, conforme citado na seção 2.4.

Buscando evitar os custos referentes ao enfileiramento das operações de escrita, para determinadas aplicações é conveniente o esquema mostrado na figura 2.2(b), no qual cada processo controla as suas próprias requisições, feitas sobre arquivos independentes, não havendo qualquer sincronização.

Para a implementação do netCDF com E/S paralela, dois projetos se destacam: o MPI-IO [IBM, 2004] e o HDF [HDF-Group, 2009a].

O MPI-IO é uma interface desenvolvida para oferecer E/S paralela ao MPI padrão [UT, 1995]. Apesar de ser usado em várias plataformas, as escritas e leituras do MPI-IO são feitas em um formato bruto, ou seja, não são oferecidas funcionalidades para manipulação de metadados que tornem o conjunto de dados autodescritivos. Também, não há garantias de portabilidade, criando dificuldades para transferência e compartilhamento de dados entre os usuários.

O HDF é um formato de arquivo, deselvolvido pela NCSA (*The National Center for Supercomputing Applications*), para armazenamento, recuperação, análise, visualização e conversão de dados científicos. Suas versões mais populares são o HDF4 [HDF-Group, 2009b] e o HDF5 [HDF-Group, 2008]. Ambas as versões armazenam matrizes multidimensionais junto com metadados, em formatos portáteis e autodescritivos. Entretanto, o HDF4 foi projetado para acesso serial aos dados, de forma muito parecida com a interface tradicional do netCDF. Já o HDF5 foi desenvolvido sobre o MPI-IO, oferecendo E/S paralela, além de outras ferramentas, como estrutura hierárquica de ar-

[3]Flags são dispositivos lógicos, expressos através de uma sequência de bits, que tem um significado e um estado (ligado ou desligado) associado a ele.

quivos, o que traz diversas opções na forma de armazenamento dos dados nos arquivos HDF5. Porém, toda essa flexibilidade traz um custo de performance associado, conforme os resultados comparativos apresentados em [Li et al., 2003].

Em virtude disso, a Northewestern University desenvolveu uma implementação paralela alternativa do netCDF, o Parallel NetCDF [Li et al., 2003], ilustrada na figura 2.2(c).

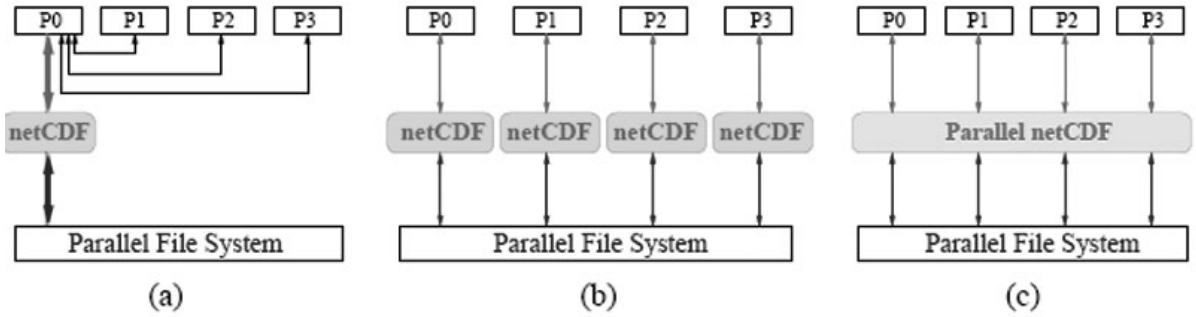

Figura 2.2: Uso do netCDF em sistemas de arquivos paralelos: (a)Enfileiramento das operações de escrita. (b)Acesso à multiplos arquivos independentes. (c)PnetCDF.

O PnetCDF foi desenvolvido sobre o MPI-IO, agregando características, como *data sieving*[4] e *two phase I/O*[5], de outros sistemas desenvolvidos em MPI-IO.

No PnetCDF um arquivo é aberto, manipulado e fechado pelos processos participantes do grupo de comunicação. Para que esses processos possam operar sobre o mesmo espaço de endereços, um grande número de modificações foi feita na API original do netCDF. São oferecidas duas APIs, uma de alto nível, em formato mais próximo ao da original, e outra chamada de API flexível, mais no estilo de programação MPI, que oferece

[4]Data sieving é uma técnica de E/S que, em vez de acessar cada porção contígua de dados separadamente, traz, para um *buffer* temporário, um bloco único e contíguo que se estende desde o primeiro byte requisitado até o último. Depois, as porções de dados requisitados são extraídas deste buffer e transferidas para um buffer acessível pelo usuário. Essa técnica foi utilizada em alguns sistemas baseados em MPI-IO, como o ROMIO [Thakur et al., 1999]

[5]*Two phase I/O*: em diversas aplicações que rodam sobre sistemas paralelos, cada processador faz muitas pequenas requisições de E/S, o que traz quedas de desempenho devido a alta latência desse tipo de operação. O *two phase I/O* consiste em coletar uma série de requisições, antes de realizá-las, formando um agregado que é analizado e recombinado. Desse processo resultam um menor número de requisições, referentes a maiores porções contíguas de dados. [Dickens and Thakur, 1998]

a possibilidade da descrição de regiões de memória não contíguas.

Outra mudança diz respeito ao conjunto de funções de acesso aos dados, dividido em dois modos distintos: coletivo e não-coletivo. As funções coletivas (identificáveis por terminarem com o sufixo "_all") envolvem, de forma sincronizada, todos os processos que estejam associados ao dataset no qual a operação será realizada, ao contrario das não-coletivas, que não tem essa restrição. O uso de operações coletivas possibilita ao PnetCDF a otimização do acesso ao dataset. As operações realizadas sobre o cabeçalho e sobre os atributos são todas coletivas.

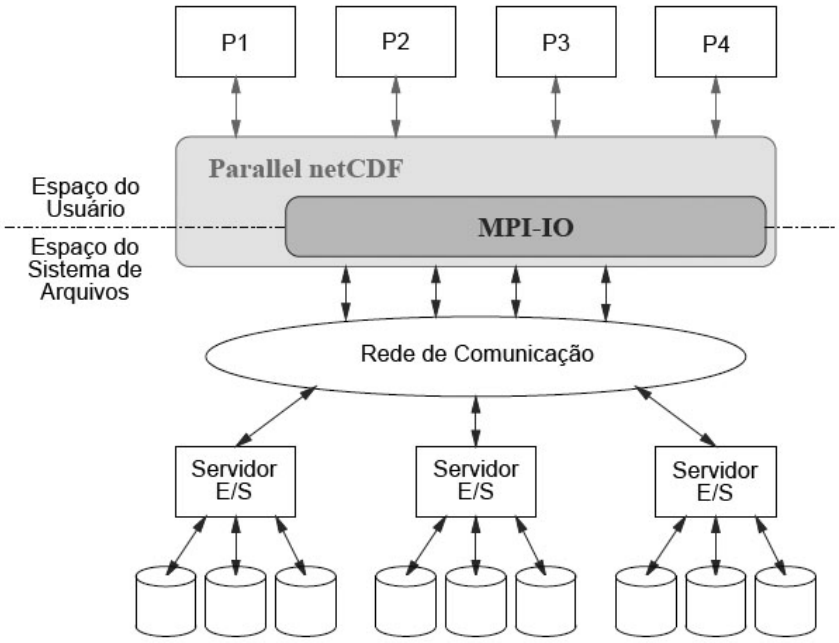

Figura 2.3: Representação da Comunicação no parallel netCDF.

A figura 2.3 ilustra, de maneira um pouco mais detalhada, o PnetCDF sobre uma arquitetura de E/S paralela. Ele roda como uma biblioteca entre o espaço do usuário e o espaço do sistema de arquivos, recebendo as requisições feitas pelos processos. Após a otimização, ele as transforma em requisições feitas através da biblioteca MPI-IO. Quando os servidores responsáveis pela E/S recebem estas últimas, eles realizam o procedimento sobre a unidade de armazenamento final.

Na versão NetCDF-4 são suportadas escritas e leituras paralelas sobre um mesmo arquivo, mediante funções de extensão da API original. O NetCDF-4 usa o modelo de

programação paralela oferecida pelo HDF5.

```
1 netcdf example_1 { // exemplo de notação CDL para um dataset netCDF
2
3 dimensions:        // nomes e tamanhos das dimensões são declarados
4                    // primeiro
5         lat = 5, lon = 10, level = 4, time = unlimited;
6 variables:         // tipos, nomes, formas e atributos das variáveis
7         float   temp(time,level,lat,lon);
8                    temp:long_name = "temperature";
9                    temp:units = "celsius";
10        float   rh(time,lat,lon);
11                   rh:long_name = "relative humidity";
12                   rh:valid_range = 0.0, 1.0; // min and max
13        int     lat(lat), lon(lon), level(level);
14                   lat:units = "degrees_north";
15                   lon:units = "degrees_east";
16                   level:units = "millibars";
17        short   time(time);
18                   time:units = "hours since 1996-1-1";
19        // atributos globais
20                   source = "Fictional Model Output";
21 data:
22        level = 1000, 850, 700, 500;
23        lat = 20, 30, 40, 50, 60;
24        lon = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
25        time = 12;
26        rh =.5,.2,.4,.2,.3,.2,.4,.5,.6,.7,
27            .1,.3,.1,.1,.1,.1,.5,.7,.8,.8,
28            .1,.2,.2,.2,.2,.5,.7,.8,.9,.9,
29            .1,.2,.3,.3,.3,.3,.7,.8,.9,.9,
30             0,.1,.2,.4,.4,.4,.4,.7,.9,.9;
31 }
```

Quadro 2.1: Exemplo de definição de um dataset escrito em CDL.

Capítulo 3

O Berkeley DataBase

Para desenvolver uma versão distribuída do netCDF, foi alterada a sua estrutura original, fazendo com que fossem utilizadas as funções oferecidas pela biblioteca do gerenciador de banco de dados Berkeley DB, em vez das primitivas do sistema operacional. Assim, esse capítulo tem o objetivo de apresentar uma descrição do Berkeley DB para que seja possível uma melhor compreensão do trabalho proposto. Primeiramente é feita uma descrição geral do sistema e das possibilidades e recursos que ele oferece e, em seguida, é feita uma explanação a respeito do modelo cliente/servidor implementado através de RPC oferecido pelo BDB e também utlizado no trabalho aqui descrito.

3.1 Um Gerenciador de Banco de Dados em Memória

Em bancos de dados tradicionais os dados residem em meios não voláteis de armazenamento, como disco rígido, e são trazidos a memória principal conforme a necessidade. Se sofrerem alguma modificação, no momento oportuno eles serão salvos no disco por um mecanismo de paginação, que grava os dados alterados no dispositivo de origem (não volátil) conforme eles precisem ser retirados da memória principal para dar lugar a outros dados requisitados, ou segundo critérios da aplicação, que executa primitivas do sistema operacional encarregadas do salvamento dos dados no meio não volátil e liberação da memória principal.

Nos bancos de dados em memória (MMDBs) a situação não é completamente diferente, invertendo-se apenas as prioridades entre memória principal e disco rígido. Nos

MMDBs, os dados residem permanentemente na memória e, devido à volatilidade deste dispositivo, podem possuir cópias de segurança no HD.

A diferença chave é que em um MMDB a cópia primária fica permanentemente na memória, o que traz importantes implicações quanto a como ela é estruturada e acessada, trazendo vantagens de desempenho para as aplicações que estiverem usando o banco de dados.

O Berkeley DB, descrito a seguir, é um gerenciador de banco de dados em memória.

3.2 Descrição Geral

O BDB [Sleepycar-Group, 2009] é um conjunto de métodos para gerenciamento de bancos de dados, fornecido sob a forma de uma biblioteca.

Duas são as possibilidades de utilizar o BDB: local ou remotamente. Devido a isso, a biblioteca de métodos que o constitui é dividida em dois subconjuntos: métodos locais e métodos remotos. É nessa perspectiva que aqui se apresenta o BDB.

3.2.1 Utilização Local

Para utilizar o BDB localmente há que se construir aplicações que invoquem métodos de gerenciamento de bancos de dados, do subconjunto de métodos locais oferecidos, e ligá-las junto a biblioteca que constitui o BDB. Portanto, as aplicações que realizam acesso aos bancos de dados compartilham espaço de endereços com o BDB (o BDB é embarcado na aplicação).

No BDB, o conceito de banco de dados corresponde a uma tabela composta de registros constituídos de dois campos: *key* e *data*. Esses campos podem possuir uma estrutura heterogênea qualquer, em conformidade com o admitido no tipo *struct* da linguagem C.

Os bancos de dados são implementados sobre os aqui denominados subespaços de endereços paginados, sendo que cada subespaço pode conter um ou mais bancos de dados. Tais subespaços são realizados sobre um *buffer pool (memory pool)*, onde cada *buffer* contém uma página de um subespaço.

Na figura 3.1 é ilustrado o esboço da organização de uma sessão de uso local do BDB, que corresponde a duas aplicações utilizando um ou mais bancos de dados, num único computador, onde se podem ver alguns conceitos até aqui apresentados, assim como outros que ainda serão introduzidos.

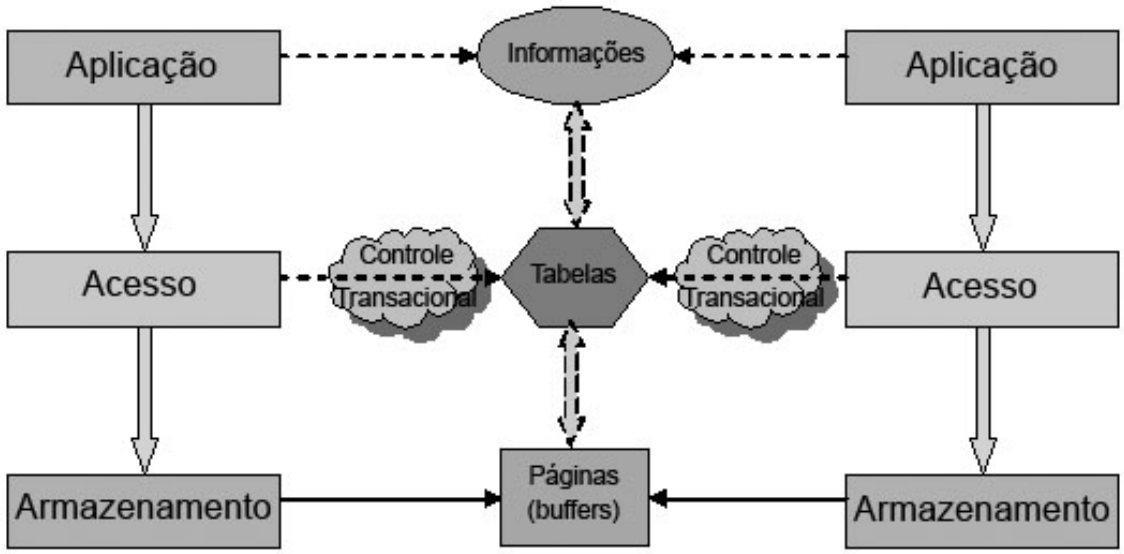

Figura 3.1: Esboço da organização de uma sessão de uso local do BDB.

Na figura, está indicado que a camada APLICAÇÃO trata as informações que lhe correspondem invocando funções que implementam métodos de acesso a bancos de dados. Como o BDB admite acesso concorrente às tabelas, através de aplicações *multithreads* e/ou *multiprocessos*, há que ser garantida a manutenção de consistência dos dados envolvidos. E isso é realizado através do conceito de transações, implementado com o uso de mecanismos de *locking, logging* e detecção de *deadlocks*. Os bloqueios de acesso, para bancos de dados estruturados sob a forma de *b-trees* ou *hash tables*, são realizados em nível de páginas.

Típicos métodos da camada ACESSO que podem ser invocadas pela camada APLICAÇÃO para manipular tabelas são:

- DB->put (*db, *txnid, *key, *data, flags);
- DB->get (*db, *txnid, *key, *data, flags);

Sendo *DB->put* um método que insere um par *key/data* no banco de dados *db*, no contexto da transação *txnid*, de acordo com especificações (aqui irrelevantes) indicadas em *flags*. O método *DB->get* recupera um par *key/data*.

Para realizarem os seus serviços, os métodos que constituem a camada ACESSO mantêm os dados pertencentes aos bancos, assim como os metadados que proporcionam a estruturação dos dados, no *memory pool*. As páginas são inseridas e recuperadas dos subespaços de armazenamento através do acionamento de métodos da camada ARMAZENAMENTO, cujo entendimento não se faz aqui necessário.

3.2.2 Utilização Remota

Para executar aplicações remotas no BDB, deve-se escolher um *host* para abrigar os bancos de dados envolvidos; instalar nesse *host* um servidor que executa métodos do subconjunto de métodos locais, quando acionado via *Remote Procedure Call (RPC)* [Srinivasan, 1995]; e instalar em *hosts* remotos as aplicações que executam os métodos do subconjunto de métodos remotos.

Ao iniciar o servidor e as aplicações, os métodos remotos executados nas aplicações invocam o servidor, via RPC, para que este acione os métodos locais a eles correspondentes.

Na figura 3.2, ilustra-se o funcionamento descrito, numa configuração em que duas aplicações estão utilizando remotamente um ou mais bancos de dados.

Como se vê na figura, os dados são armazenados sob a forma de páginas no *host* servidor, da mesma forma como descrito para o caso da utilização local, portanto centralizados num único computador.

Os métodos remotos possuem a mesma assinatura dos métodos locais, e são selecionados ao invés destes pela indicação, em uma *flag* de controle, que se tratam de invocações remotas. Tais métodos são construídos de forma a invocar seus correspondentes métodos locais no host servidor. Esse mecanismo é implementado através do paradigma RPC.

Pode-se então referir que, mesmo no caso de utilização remota, o BDB gerencia as tabelas e os subespaços de armazenamento de modo centralizado. Na verdade, ele é organizado de tal forma que um servidor, executando localmente no âmbito das aplicações remotas, invoca os métodos locais correspondentes aos métodos remotos invocados nas

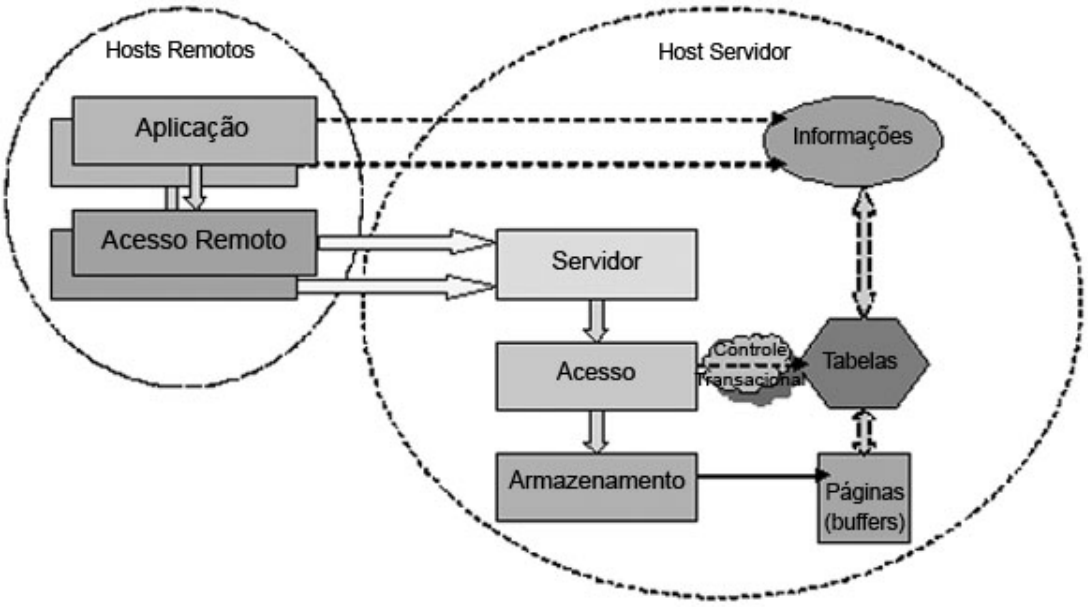

Figura 3.2: Ilustração do acionamento remoto do BDB por duas aplicações.

aplicações.

Devido ao possível acesso compartilhado das tabelas entre o servidor e outras aplicações locais (situação não representada na figura por motivo de clareza), na utilização remota também é oferecida a possibilidade de manutenção de consistência dos dados através de controle transacional. Isso é realizado da mesma forma que no caso de utilização local, pela própria estrutura funcional escolhida para o BDB, conforme descrito.

No entanto, como os mecanismos de manutenção de consistência são implementados na camada aqui denominada ACESSO, apenas os métodos dessa camada podem ser acionados remotamente.

Capítulo 4

NetCDF sobre o Berkeley DB

O MPnetCDF é um sistema distribuído desenvolvido com o objetivo de testar a idéia proposta neste trabalho: desenvolver uma camada intermediária, capaz de distribuir os dados da aplicação – NetCDF – entre os servidores que constituem o gerenciador de armazenamento, composto por instâncias de um sistema de gerenciamento de bancos de dados em memória – BDB – instalados nos nodos de uma rede de computadores – *cluster* – criando, assim, um espaço de armazenamento compartilhado e voltado para as necessidades da aplicação. Dessa maneira, eliminam-se os custos para manutenção da consistência presentes em outras alternativas genéricas, com menor esforço de implementação.

Ele foi desenvolvido a partir da versão clássica do netCDF, que foi modificada para passar a operar sobre as funções oferecidas pelo BDB, em vez das primitivas do sistema operacional. Essa versão foi escolhida por apresentar maior compatibilidade com o Berkeley DB como, por exemplo, quanto ao endereçamento.

Ao longo desse capítulo serão apresentadas as razões pelas quais escolhemos o BDB como sistema base para distribuição do netCDF.

4.1 O netCDF pode ser implementado sobre um Database Management System?

A maioria dos sistemas gerenciadores de bancos de dados mostram-se inadequados para armazenar dados na forma de matrizes, como faz o NetCDF. Problemas comuns

são a incapacidade de DBMSs que seguem o modelo relacional[1] de gerenciar objetos multidimensionais, apresentando dificuldades de organizar e indexar os dados de maneira eficiente, o que gera problemas de performance.

Entretanto, conforme explicado na seção 3.2.1, o Berkeley DB gerencia tabelas compostas de dois campos básicos, chave e valor. Ambos podem ser formados por estruturas de dados complexas (dependendo do método de acesso utilizado), como matrizes multidimendionais. Assim, no BDB as matrizes não precisam ser representadas na forma de relações. Na versão NetCDF descrita nesse trabalho, elas são armazenadas inteiramente no campo valor. Isso não traz restrição na indexação dos dados, ou seja, um determinado elemento da matriz pode ser acessado diretamente, sem a necessidade de acessá-la toda e depois percorrê-la, elemento a elemento. Isso, porque tendo os metadados referentes aos índices e aos tamanho dos elementos, é possível calcular a posição exata na memória do elemento desejado.

O BDB também oferece alguns métodos de acesso que o tornam adaptável a qualquer aplicação e um motor de busca bastante eficiente. Quase todos os recursos e subsistemas que o compõem (controle transacional, detecção de *deadlocks*, subsistema de recuperação em caso de falhas, etc.) podem ser habilitados ou não, a escolha do usuário, que pode assim adequá-lo a sua aplicação.

Assim, o BDB mostra-se mais flexível e adaptável quando comparado a a gererenciadores de bancos de dados tradicionais, o que permite utilizá-lo como sistema base para o netCDF, podendo organizar os dados (matrizes) de maneira apropriada e eficaz (ver resultados no capítulo 5).

4.2 Distribuição dos Dados

No BDB, um banco de dados é uma tabela. Vários bancos de dados podem ser encapsulados em um *environment*, um ambiente utilizado para possibilitar acesso concorrente entre vários processos e/ou threads. Supõe-se que as aplicações que estiverem

[1] O modelo relacional é um modelo de dados cuja definição é baseada em lógica de predicados e teoria de conjuntos. Seu princípio básico é que todos os dados estão guardados em tabelas ou, matematicamente falando, relações.

compartilhando os bancos de dados em um *environment* "confiam" umas nas outras. Ou seja, os acessos compartilhados não acarretam problemas de consistência. Os *environments* são também envolvidos na comunicação entre cliente e servidor, que só pode ser realizada através deles.

Cada nodo que está executando um servidor BDB espera pelas solicitações remotas que partem da aplicação. Os servidores mantêm um *environment* cada, que reúne seu conjunto de bancos de dados. Por motivos de simplificação do projeto e procurando oferecer aos datasets maiores possibilidades de expansão quanto ao volume de dados, ao fazer a adaptação entre os sistemas, decidiu-se que cada dataset (arquivo NetCDF) ficaria armazenado em um banco de dados (arquivo BDB) e que cada banco de dados ficaria fisicamente armazenado em um arquivo direferente. Isso porque, como existe um tamanho máximo de arquivo manipulável pelo BDB, se fossem colocados, por exemplo, dois bancos de dados, ou seja, dois datasets, em um mesmo arquivo, eles ficariam com suas capacidades de armazenamento reduzidas à metade. Na seção 4.3, abordaremos os limites de armazenamento.

Quando a aplicação solicita a criação de um dataset, ao passar pela camada de distribuição, é determinado em qual servidor será criado o arquivo. Se não houver determinação explicita por parte da aplicação (ver seção 4.4), essa escolha é feita automaticamente.

Para possibilitar futuras operações sobre o dataset, é mantida uma **tabela de distribuição**, que relaciona o nome do arquivo ao servidor no qual ele foi criado. Por exemplo, se o dataset de nome "modelo_climatico_2009" for criado no servidor de endereço 192.168.135, essa informação será armazenada como uma entrada na tabela de distribuição, que será consultada a cada nova operação – inserções e consultas, por exemplo – que for gerada para o mesmo dataset. Uma representação simplificada desse procedimento é mostrada na figura 4.1.

No MPnetCDF cada arquivo fica inteiramente em um mesmo nodo. Ou seja, todas as requisições referentes a um mesmo dataset são enviadas para o mesmo nodo, diferentemente das outras versões paralelas (PnetCDF e NetCDF-4/HDF5), nas quais os datasets ficam divididos entre os nodos, de forma que as suas informações ficam espalhadas entre eles; para reunir todo o dataset são necessárias buscas em todos os nodos participantes.

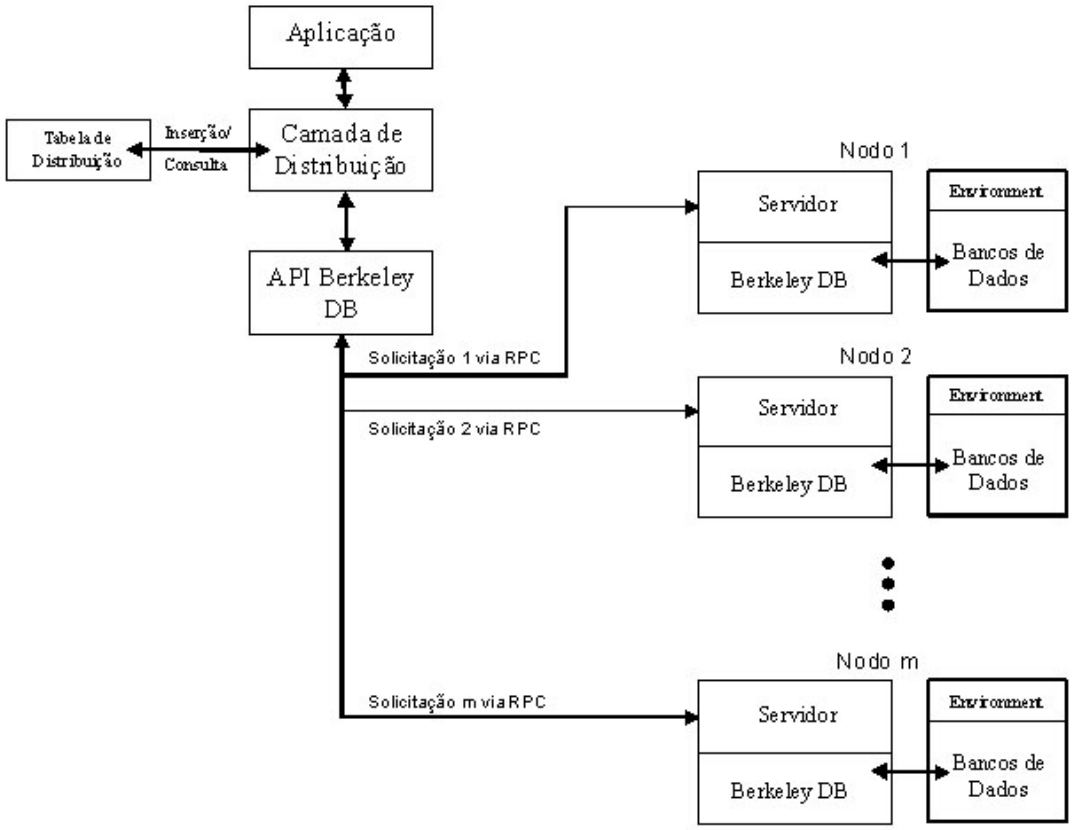

Figura 4.1: Solicitações da aplicação, no MPnetCDF, passam pela camada de distribuição, onde é consultada a tabela de distribuição, e seguem para o seu servidor destino.

Como o cabeçalho do arquivo netCDF contém todas as informações que tornam o dataset autodescritivo, são-lhe necessárias consultas constantes, podendo vir ele a tornar-se um gargalo no sistema. Cada uma das versões paralelas adota um método diferente para resolver a questão.

O PnetCDF mantém uma cópia do cabeçalho em cada nodo participante, resolvendo, assim, a questão do gargalo, porém surge uma dificuladade devido à necessidade de sincronização dos cabeçalhos, em todos os nodos, quando no modo de definição (ver seção 2.3), ou seja, modificações no cabeçalho de um nodo precisam ser replicadas nos demais.

Os dados ficam divididos entre os processos em subespaços contíguos em uma ordem predefinida, para variáveis de tamanho fixo, ou intercalados por registro seguindo um padrão regular, para variáveis de registro. O padrão de acesso aos dados é representado

como uma *visão*, que é um subconjunto dos dados acessado através de um descritor de arquivo. Cada processo tem uma visão diferente dos dados. Através das funções coletivas (ver seção 2.5), todos os processos, em conjunto, podem realizar a transferência de grandes porções contíguas de dados, que estejam no domínio de diferentes processos.

No netCDF-4/HDF5 os metadados referentes a um objeto (variável, dimensão ou atributo) ficam espalhados entre os nodos em pequenos blocos de cabeçalho associados aos objetos (porções de dados divididas entre os processos). Com isso, para operar sobre um objeto, é necessário uma busca para encontrar as informações referentes a ele e, então, acessá-lo. Além dos custos óbvios para localizar e acessar tais informações, ao fazer possíveis modificações nos metadados é necessário atualizar as cópias dos respectivos blocos, que estejam na memória de outros nodos.

No MPnetCDF, como cada nodo cuida de um ou mais datasets inteiros, não há necessidade nenhuma de sincronização, já que o conteúdo do cabeçalho de um dataset não interessa ao outro. Assim, cada cabeçalho está sempre na árvore do seu dataset.

Além das questões referentes a sincronização, o propósito de tornar o MPnetCDF compatível e robusto frente às aplicações já desenvolvidas para o netCDF clássico é um outro aspecto que levou a opção pela distribuição dos dados em nível de dataset entre os nodos e não por uma granulosidade maior, como a distribuição das variáveis. A API definida pelo NetCDF foi integralmente implementada de modo que qualquer aplicação desenvolvida para a versão clássica do NetCDF execute sem qualquer alteração. Entretanto, para oferecer ao usuário uma opção de controle sobre a distribuição dos dados, foram adicionadas duas novas funções a API original, que estão definidas na seção 4.4.

4.3 Estrutura do Arquivo MPnetCDF

Embora a estrutura do arquivo netCDF permaneça, após ter sido implementada sobre o Berkeley DB, conceitualmente dividida em cabeçalho e área de dados, a forma de organização dos dados e metadados foi modificada.

No Berkeley DB, por padrão, cada tabela é, fisicamente, armazenada em um arquivo. Assim, ao fazer a adapção entre os sistemas, foi estabelecido que cada dataset (arquivo netCDF) ficaria armazenado em uma tabela (arquivo BDB).

Os dados nas tabelas ficam organizados segundo um dos métodos de acesso oferecidos pela interface do BDB. Ao passar a usá-lo como gerenciador de dados, foi escolhido o método de acesso **B-tree** ou **árvore B**.

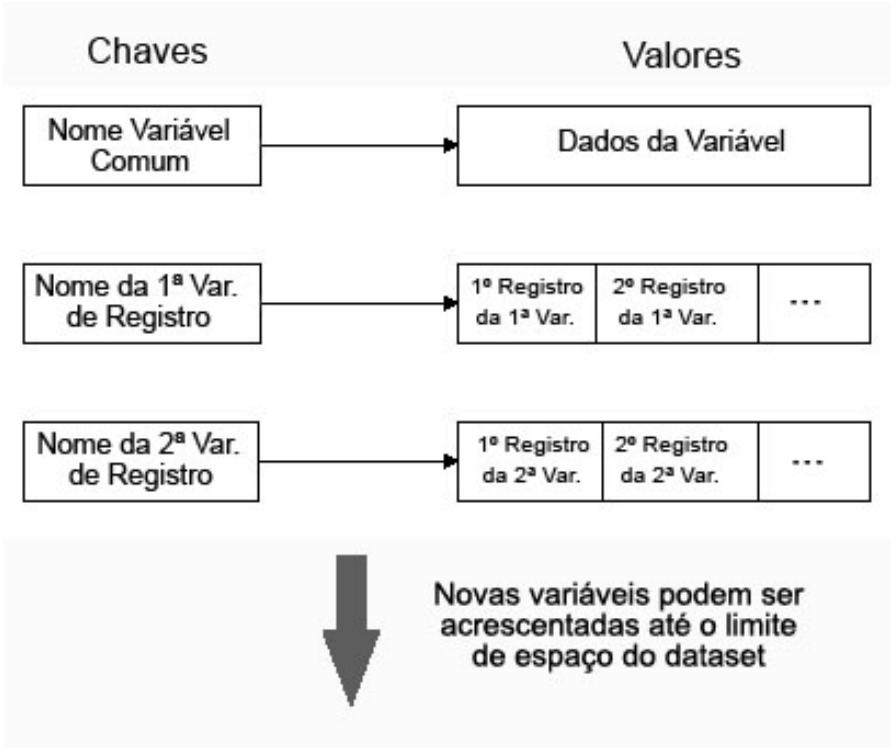

Figura 4.2: Estrutura das variáveis do MPnetCDF.

Árvores B são estruturas de dados que apresentam certas características que lhes dão vantagem sobre outros tipos de árvores, principalmente no que se refere ao armazento externo de dados, como em discos magnéticos. Por serem estruturas de dados balanceadas, o número máximo de nós acessados para encontrar uma chave se mantém pequeno. Outra caracterítica que faz com que sejam usadas frequentemente em sistemas de arquivos reais é que, devido às suas propriedades, muito pouco espaço de armazenamento é desperdiçado ([Tenenbaum et al., 1995]).

As árvores B, por se ajustarem e balancearem a cada operação, também têm vantagens em relação as tabelas *hash* para inserção e exclusão de dados. Na busca são equivalentes, mas levam vantagem se o sistema faz muitas requisições de chaves subsequentes.

Essa forma de armazenamento acarreta na liberação de algumas restrições apre-

sentadas originalmente pelo netCDF. Conforme referido na seção 2.4, para qualquer modificação no dataset que leve ao aumento ou diminuição do cabeçalho, ou da área de dados de tamanho fixo, era necessário a cópia dos dados do arquivo para uma nova área de armazenamento que fosse compatível com o novo tamanho. No MPnetCDF isso não acontece porque as chaves, do par chave/valor, ficam ordenadas em forma de uma árvore B. Assim, uma inserção ou uma exclusão na estrutura fazem com que um novo nó seja acrescentado ou retirado da árvore, que se ajusta, de forma balanceada, através de um "jogo de ponteiros" ou seja, mudam apenas os endereços das áreas apontadas pelos nodos que restarem na árvore.

A figura 4.3 ilustra o formato final de uma árvore B de ordem um, resultante da criação de um dataset que descreve um sistema que faz leituras de temperatura e pressão, a cada intervalo de tempo, em regiões do espaço determinadas em função de altitude, longitude e latitude. Tempo, altitude, longitude e latitude são variáveis de coordenada (que fazem o mapeamento entre os índices da matriz e os valores das coordenadas, conforme descrito na seção 2.3.2). Header é o cabeçalho do dataset. Note-se que ele é tratado da mesma forma que as demais variáveis e que as chaves ficam ordenadas em ordem alfabética crescente.

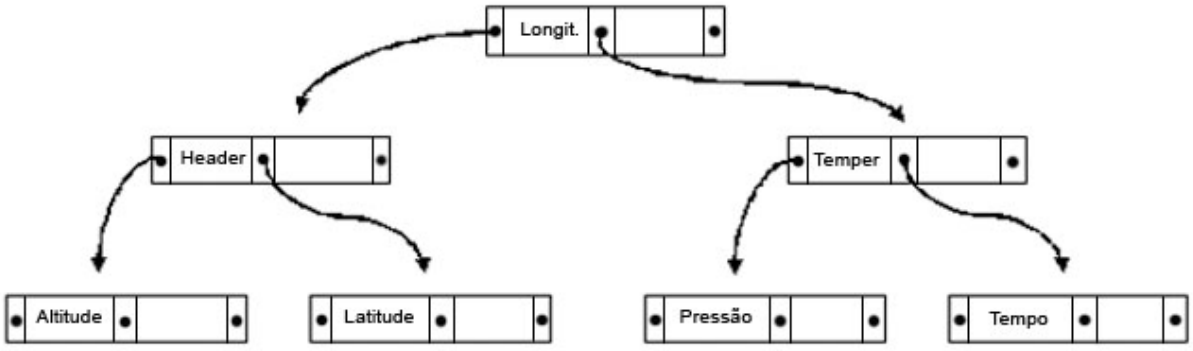

Figura 4.3: Representação de uma árvore B de ordem 1 de um dataset, com seis variáveis, incuindo cabeçalho.

Ao compararmos a figura 4.2 com a figura 2.1, na qual vimos que todos os dados e metadados presentes no arquivo netCDF eram, originalmente, armazenados de forma sequencial, percebemos que o modo de organização é bastante diferente.

Vemos que os conteúdos das variáveis sempre ocupam o campo valor e são referenciados pelo campo chave, indexado pelo nome da variável. As variáveis comuns (que incluem o cabeçalho), de tamanho fixo, simplesmente apontam para o endereço do primeiro byte do conjunto de informações ao qual se referem. As variáveis de registros não seguem o esquema de armazenamento contíguo intercalado utilizado para as variáveis desse tipo no NetCDF clássico. Cada variável aponta para seu próprio conjunto de registros e elas são independentes entre si, ou seja, seus conteúdos ficam contiguamante armazenados, porém separados dos conteúdos das demais. Note-se que as chaves na figura 4.2 não estão dispostas em formato de B-tree por motivo de clareza.

Novas variáveis podem ser adicionadas até o limite de armazenamento do dataset, que pode ser de até 256 Terabytes, e cada campo, chave ou valor, pode ter até 4 Gigabytes. Esses tamanhos são comparáveis aos da versão 64 bits do netCDF. No netCDF clássico o tamanho máximo é de 2 GigaBytes por arquivo, o que pode ser pouco para alguns parâmetros atuais de volume dados. Já no netCDF-4/HDF5 os arquivos podem ser tão grandes quanto o sistema de arquivos permitir.

Uma vantagem, advinda dessa forma de organizar os dados e metadados de que tratamos até aqui, é descrita a seguir. O netCDF-3 permite apenas a declaração de uma única dimensão ilimitada, conforme visto na seção 2.3.1; para muitas aplicações essa dimensão é o tempo. Imaginemos, porém, uma aplicação que faz coleta de vários tipos de dados com frequências diferentes. Para melhor compreensão voltaremos ao exemplo da seção 2.3.4. Consideremos que a temperatura precisasse ser medida a cada trinta minutos e a humidade relativa apenas a cada noventa minutos. Esse tipo de situação não deve ser raro, dado que algumas variáveis, especialmente na natureza, mudam de valor mais rapidamente que outras. Para que essa situação pudesse ser modelada pelo netCDF, idealmente seriam necessárias a declaração de duas dimensões ilimitadas, uma para cada frequência. Como isso não é possível, duas soluções são possíveis:

Na primeira, declarar-se-ia apenas uma dimensão ilimitada, aquela que variasse mais rapidamente, de forma que as outras variáveis ficassem preenchidas por algum elemento que representasse um valor nulo para os tempos em que elas não fossem levantadas. No nosso exemplo, para a primeira amostra, no início da aplicação, teríamos um valor de temperatura e humidade relativa válidos; para a segunda e terceira amostras, decorridos

trinta e sessenta minutos, respectivamente, após o início do experimento, teríamos apenas valores de temperatura válidos e nulos para a humidade relativa, e assim sucessivamente. Essa opção tem duas desvantagens: a de desperdiçar tempo de processamento e de utilizar espaço para armazenamentos desnecessários, deixando "buracos" de valores nulos no dataset. Esse problema pode ser melhor visualisado observando-se a figura 2.1.

A segunda opção, que deve ser a mais comumente utlizada pelos usuários netCDF, seria a criação de tantos datasets quantos forem as dimensões ilimitadas. Esta opção traria um overhead bem menor, na replicação dos dados das dimensões normais, se comparada ao método anterior, cujo overhead fica multiplicado pelo número de amostras e de variáveis de registro no modelo.

Para novas aplicações que utilizem o MPnetCDF, podem ser declaradas várias dimensões ilimitadas em um mesmo dataset e as váriaveis de registro podem variar, cada uma, em função de uma (e apenas uma) dimensão ilimitada diferente, o que soluciona o problema descrito acima.

4.4 Extensão da API NetCDF

O conjunto de funções originais oferecidas pela interface netCDF foi integralmente mantido na nossa versão, de forma que qualquer aplicação desenvolvida para a versão clássica do netCDF poderá executar sem qualquer alteração. Entretanto, para oferecer ao usuário maior arbitragem sobre alguns novos parâmetros, adicionamos duas novas funções, opcionais, a API original.

A primeira função surge devido à necessidade de informarmos ao MPnetCDF quais *hosts* poderão ser usados para armazenar os datasets. Essa informação é passada na forma de uma lista de IPs dentro de um arquivo texto. Cada IP deve ocupar uma linha, conforme mostrado no exemplo abaixo:

```
192.168.1.35
192.168.1.43
192.168.1.109
```

Neste exemplo, seriam usados três *hosts* da rede interna, cada um especificado numa linha.

O arquivo texto contendo a lista de IPs deve, por padrão, ser criado no mesmo diretório de onde é executada a aplicação, e ter o nome **"servs"**. Se esse procedimento for adotado, nenhuma alteração precisa ser feita na aplicação.

Caso o usuário deseje indicar outro caminho e nome para o arquivo, ele poderá, então, utilizar a seguinte função:

int nc_set_iparchname(char *path)

O único argumento, **path**, indica o caminho (em formato unix) e o nome do arquivo que contém a lista. É retornado um valor diferente de zero caso o arquivo não exista no caminho especificado e zero em caso de sucesso. A função deve ser chamada antes do primeiro dataset ser criado, ou seja, antes de ser chamada pela primeirva vez a função nc_create(...) ou nc__create(...), que são funções responsáveis pela criação do dataset (para mais detalhes sobre essas funções, consultar "The NetCDF C Interface Guide" [Rew et al., 2008a]). No quadro 4.1 é apresentado um exemplo.

```
int main()
{
     ... //Declarações e definições
     ret = nc_set_iparchname("/home/user/lista");
     ...
     ret = nc_create(...); //Criação do dataset
     ... // Criação das dimensões, variaveis, atributos e gravação dos dados
     ret = nc_close(...); //Encerramento do dataset
     return 0;
}
```

Quadro 4.1: Definição do caminho do arquivo contendo a lista de servidores.

Para saber onde criar o dataset, será consultado o arquivo de nome "lista" e o caminho onde ele será procurado será "/home/user/".

A segunda função oferece a possibilidade de indicar em qual dos computadores listados no arquivo de servidores o dataset seguinte será criado.

Continuando com o primeiro exemplo apresentado acima, se houver três servidores disponíveis e pretende-se criar quatro datasets, por padrão eles serão criados na ordem em que aparecem no arquivo. Ou seja, o primeiro dataset será criado no servidor que tem seu IP na primeira linha do arquivo, o segundo no servidor com IP na segunda linha, e assim por diante, de forma cíclica.

Contudo, dependendo da aplicação, esse esquema circular de distribuição de dados pode não ser o ideal. Assim, quando for desejada a criação de um dataset em um servidor específico, deve-se usar a função:

int nc_set_serv(char *ip)

Seu único argumento, **ip**, deve conter um endereço IP válido, ou seja, que esteja presente na lista do arquivo de servidores. A função retorna zero em caso de sucesso e outro valor em caso de erro.

Ela deve ser chamada antes da chamada de cada nc_create(...) ou nc__create(...), para que o dataset correspondente seja direcionado para o servidor indicado. No quadro 4.2 apresenta-se um exemplo.

Nesse quadro, podemos perceber que o primeiro dataset é criado no primeiro servidor, porque não há indicação explicita direcionando a chamada. Já o segundo dataset é direcionado também ao primeiro servidor. Como não há novas indicações para os outros dois datasets, eles seguem o esquema circular de armazenamento a partir do último endereço onde houve criação.

Essa função é importante para que o usuário possa encontrar a melhor distribuição dos dados na rede. Cada aplicação tem uma configuração mais propícia na distribuição dos dados para obter um melhor desempenho. Assim, o usuário pode especificar quais datasets devem ficar no mesmo nodo e quais devem ficar separados. Por exemplo, um conjunto de três arquivos que sejam menos acessados pela aplicação podem ficar juntos em um mesmo nodo, enquanto outro dataset, que seja mais comumente acessado, pode ficar isolado em um outro servidor.

```
int main()
{
     ... //Declarações e definições

     ret = nc_create(...); //Criação do 1º dataset no 1º servidor
     ... // Criação das dimensões, variaveis, atributos e gravação dos dados
     ret = nc_close(...); //Encerramento do 1º dataset

     ret = nc_set_serv("192.168.1.35");
     ...
     ret = nc_create(...); //Criação do 2º dataset no 1º servidor
     ... // Criação das dimensões, variaveis, atributos e gravação dos dados
     ret = nc_close(...); //Encerramento do 2º dataset

     ret = nc_create(...); //Criação do 3º dataset no 2º servidor
     ... // Criação das dimensões, variaveis, atributos e gravação dos dados
     ret = nc_close(...); //Encerramento do 3º dataset

     ret = nc_create(...); //Criação do 4º dataset no 3º servidor
     ... // Criação das dimensões, variaveis, atributos e gravação dos dados
     ret = nc_close(...); //Encerramento do 4º dataset

     return 0;
}
```

Quadro 4.2: Exemplo de distribuição de datasets entre os nodos do cluster através da função nc_set_serv().

Capítulo 5

Testes e Resultados

Neste capítulo, analisa-se o desempenho do MPnetCDF frente a outras versões. Inicialmente é feita uma comparação com uma versão distribuída do netCDF, o PnetCDF. Em seguida, são apresentados os resultados obtidos da comparação com a versão netCDF clássica, centralizada.

Embora tenha sido citado no trabalho a existência de outra versão distribuída do netCDF, o NetCDF-4/HDF5, os resultados apresentados no artigo [Li et al., 2003], que a compara com o PnetCDF, indicam melhor desempenho deste último. Em virtude disso, escolheu-se apenas este para comparação.

Ao todo, são apresentados resultados de três classes de comparação:

- MPnetCDF distribuído X PnetCDF
- MPnetCDF distribuído X netCDF
- MPnetCDF centralizado X netCDF

Todos os testes foram feitos no cluster de computadores do Núcleo de Computação Científica da FURG. constituído por dez computadores: um front-end, um master e oito slaves. Os testes foram feitos nos slaves, cada um com dois processadores AMD Opteron 265 Dual Core, 3Gb de memória Ram, 80Gb de espaço em HD e duas placas de rede Gigabit, conectados a um switch gigabit, exclusivo para essa finalidade.

Para obtenção dos resultados foi utilizado um *benchmark* que acompanha o pacote PnetCDF e está escrito de modo a executar sobre quatro processadores. Foi desenvolvida uma versão similar, compatível com o MPnetCDF e netCDF clássico.

O dataset do teste é composto de quatro dimensões, x, y, z e time, sendo esta última ilimitada, e de quatro variáveis, duas delas, square(x,y) e cube(x,y,z), são variáveis comuns, ou seja, não apresentam a dimensão ilimitada, e duas são variáveis de registro, time(time), que é uma variável de coordenada (está em função de apenas uma dimensão e leva o seu nome), e xytime(time,x,y). Todas armazenam números de ponto flutuante (float).

Apesar de não ter sido desenvolvido com o objetivo de descrever um situação física real, o teste foi escolhido por utilizar todos os tipos de variáveis oferecidas pelo netCDF, oportunizando a avaliação do comportamento dos sistemas ao tratar variáveis unidimensionais e multidimensionais, registros e possibilidade de leitura pontual dentro de variáveis.

No benchmark original são atribuídas às dimensões x, y e z tamanhos iguais a 100 elementos. Para avaliar o desempenho dos sistemas foram realizados testes com x=y=z=60, 80, 100 e 120. Em todos os casos o número de registros gravados em time e xytime foi 100.

5.1 MPnetCDF X PnetCDF

Para analisar o comportamento do MPnetCDF, ele foi executado sobre quatro nodos do cluster. Foram medidos os tempos de escrita e leitura dos dados para tamanhos diferentes de matrizes, conforme descrito anteriormente.

Como foram encontradas diferenças nas medições para cada uma das execuções, foram coletados tempos de cinquenta repetições e calculada a média, o intervalo de confiança (IC) de 95% e o desvio padrão das amostras, tanto para operação d eescrita e leitura. O IC indica que há 95% de certeza de que a média se encontre no intervalo calculado.

Foram utilizados quatro nodos porque o *benchmark* que acompanha o pacote do PnetCDF foi desenvolvido para essa configuração, conforme já foi dito. Além disso, entendeu-se desnecessária a avaliação com um maior número de nodos pois, devido a forma de distribuição, é pouco provável que os resultados sejam diferentes dos encontrados.

A tabela 5.1 e a figura 5.1 mostram os resultados obtidos para operações de escrita.

Nelas se podem observar tempos médios sensívelmente menores no MPnetCDF.

Tabela 5.1: Valores obtidos para escrita.

x,y,z	Sistema	Tempo Médio (ms)	Desv. Padr.	IC
60	PnetCDF	117,00	120,19	± 33,31
	MPnetCDF	21,60	5,10	± 1,41
80	PnetCDF	161,00	119,66	± 33,17
	MPnetCDF	44,00	6,39	± 1,77
100	PnetCDF	264,80	175,07	± 48,53
	MPnetCDF	68,60	8,08	± 2,24
120	PnetCDF	***	***	***
	MPnetCDF	102,60	9,43	± 2,62

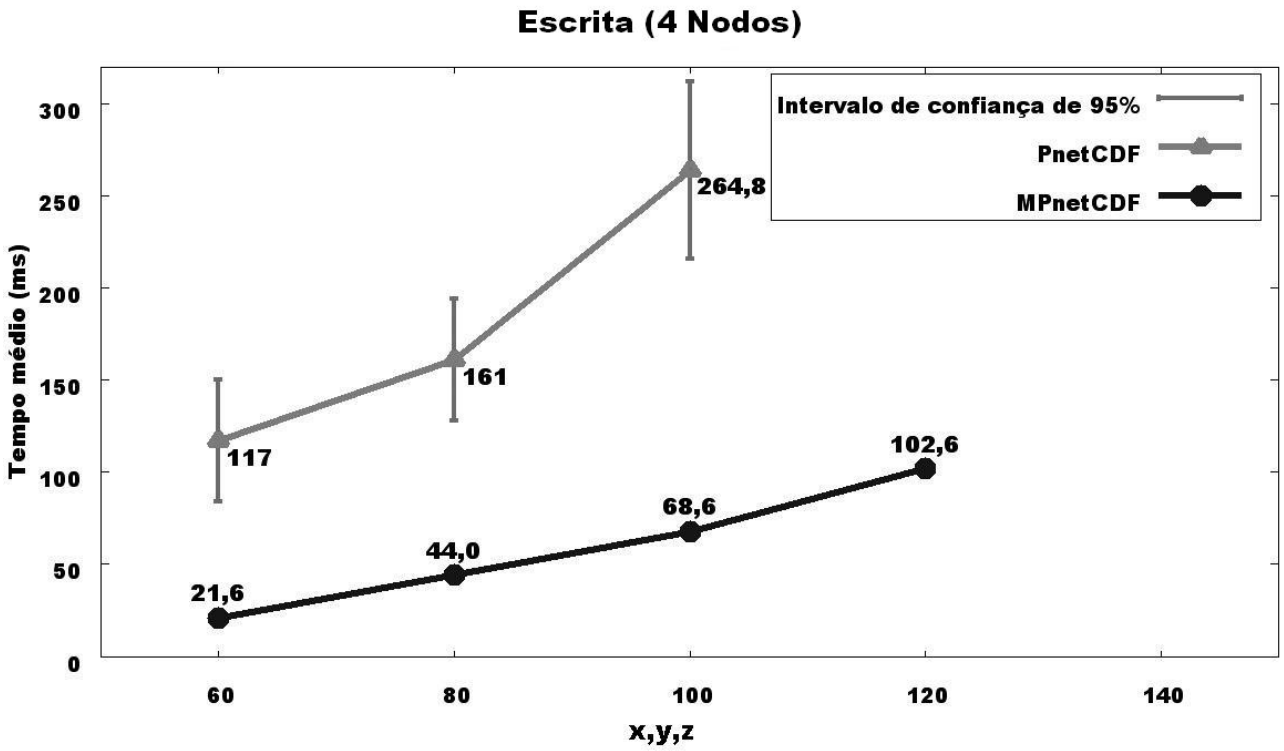

Figura 5.1: Gráfico com os tempos médios de escrita do PnetCDF e do MPnetCDF.

Isso se deve, principalmente, a ele ser um sistema em memória que, como já foi dito, não realiza gravação dos dados em disco rígido. Parte dessa diferença também é devida às características do MPI (paradigma sobre o qual está implementada a comunicação em rede do PnetCDF), o qual, conforme descrito em [Qureshi and Rashid, 2005], apresenta maiores latências (roundtrip dos pacotes de dados entre cliente e servidor) e menores larguras de banda de passagem quando comparado ao RPC (paradigma sobre o qual está implementada a comunicação em rede do MPnetCDF).

Os desvios padrão em torno da média também são sensívelmente menores, mostrando que o MPnetCDF tem um comportamento mais estável. Como as condições de testes foram rigorosamente as mesmas para ambos os sistemas (mesmo ambiente e única tarefa em execução no momento dos testes), as diferenças entre medições obtidas no PnetCDF devem ocorrer também em função de algumas características do MPI, que é estruturado em três camadas, cada uma com diversas tarefas, onde muitas decisões são tomadas em tempo de execução [Open-MPI, 2009], diferentemente do RPC, que tem um modelo de execução mais simples.

Além disso, sua camada inferior é uma camada de convergência entre o sistema operacional e a camada superior. Nos sistemas operacionais tipo unix, normalmente essa camada utiliza o protocolo SSH (Security Shell) [SSH, 2009], bem mais pesado do que o UDP.

Note-se que os valores do PnetCDF para matrizes de tamanho cento e vinte não foram calculados porque esse número excede o limite máximo de dimensão suportado por esse sistema.

No gráfico, as barras verticais representam o intervalo de confiança do tempo médio encontrado, com um grau de certeza de noventa e cinco por cento. Por exemplo, para matrizes com dimensões de tamanho sessenta tem-se um tempo médio de 117 ms e um intervalo de confiança de $\pm$ 33,31 ms, ou seja, podemos afirmar, com noventa e cinco por cento de certeza, que a média dos tempos de execução encontra-se no intervalo entre 83,69 ms e 150,31 ms. Da mesma forma, para o MPnetCDF temos, ainda para matrizes com dimensões de tamanho sessenta, um tempo médio de 21,6 ms e um IC de $\pm$ 1,41 ms, ou seja, podemos afirmar com 95% de certeza que a média dos tempos de execução

Tabela 5.2: Valores obtidos para leitura.

x,y,z	Sistema	Tempo Médio (ms)	Desv. Padr.	IC
60	PnetCDF	184,60	114,52	± 31,74
	MPnetCDF	34,80	6,46	± 1,79
80	PnetCDF	191,60	129,07	± 35,78
	MPnetCDF	73,20	10,19	± 2,82
100	PnetCDF	210,80	135,54	± 37,57
	MPnetCDF	113,80	9,87	± 2,74
120	PnetCDF	***	***	***
	MPnetCDF	179,80	14,64	± 4,06

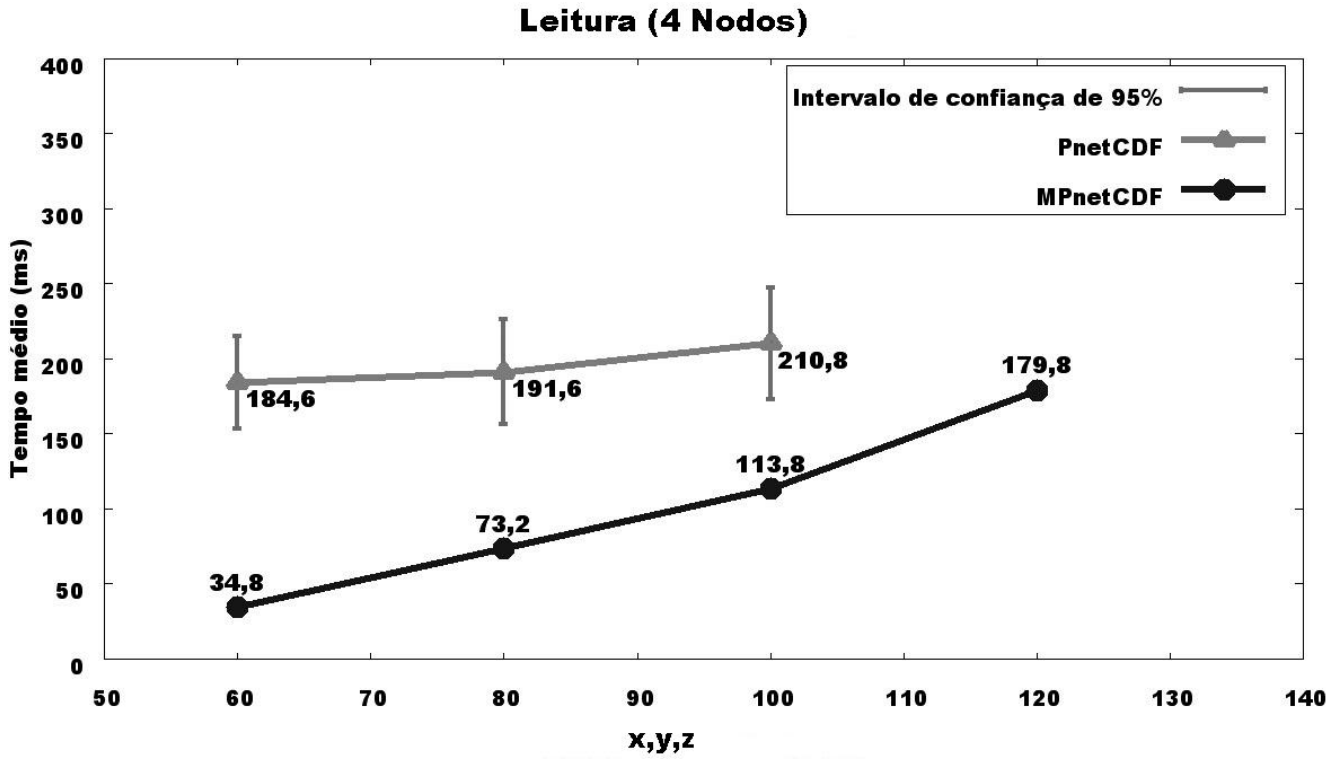

Figura 5.2: Gráfico com os tempos médios de leitura do PnetCDF e do MPnetCDF.

encontra-se no intervalo entre 20,19 ms e 23,01 ms e assim sucessivamente[1]. Disso se pode deparar que o MPnetCDF é também mais estável que o PnetCDF, no que diz respeito aos tempos de escrita.

Os resultados mostrados na tabela 5.2 e na figura 5.2 são referentes aos tempos de leitura, onde também se percebe uma diferença significativa em favor do MPnetCDF no que diz respeito ao desempenho e estabilidade. Contudo, diferentemente do que ocorreu na escrita, as curvas apresentadas tendem a se aproximarem a medida que o tamanho das dimensões das matrizes aumenta.

5.2 MPnetCDF (4 Nodos) X NetCDF

Nesta seção são apresentados os resultados da comparação entre o MPnetCDF executando sobre 4 nodos do cluster e a versão NetCDF clássica, que executa de forma centralizada.

A tabela 5.3 apresenta os resultados dos tempos medidos na execução do NetCDF para a escrita, repetindo os resultados apresentados na sessão anterior para o MPnetCDF, de forma que possam ser mais facilmente comparados.

Como os dados no MPnetCDF são armazenados na memória volátil e no NetCDF o são no disco rígido, era esperado que a escrita do primeiro fosse mais rápida. Porém como os tempos medidos envolveram a inicialização do sistema (start up), o que, para o MPnetCDF corresponde a conectar aos servidores, criar os datasets nos nodos, etc e como um nodo só é inicializado após o outro, o tempo para realização das operações fica aumentado. Futuramente, será realizada modificação no sistema para que a inicialização de todos os nodos ocorra ao mesmo tempo, o que deverá minimizar essa diferença.

Da mesma forma, na leitura, cujos resultados são mostrados na tabela 5.4 e na figura 5.4, os valores obtidos foram os esperados. Os tempos medidos envolveram a inicialização do sistema (conexão aos servidores, abertura dos datasets, etc.) e a consulta a tabela de distribuição (ver seção 4.2) para descobrir em qual nodo estão os dados a serem buscados. Por outro lado, o NetCDF, que escreve os dados de forma sequêncial, apenas

[1] As barras referentes aos intervalos de confiança não aparecem para alguns pontos dos gráficos devido à escala de representação.

Tabela 5.3: Resultados dos tempos de escrita do NetCDF e do MPnetCDF (4 nodos).

x,y,z	Sistema	Tempo Médio (ms)	Desv. Padr.	IC
60	NetCDF	22,41	5,91	± 1,64
	MPnetCDF (4 Nodos)	21,60	5,10	± 1,41
80	NetCDF	31,42	11,07	± 3,07
	MPnetCDF (4 Nodos)	44,00	6,39	± 1,77
100	NetCDF	48,83	13,80	± 3,83
	MPnetCDF (4 Nodos)	68,60	8,08	± 2,24
120	NetCDF	65,65	15,55	± 4,31
	MPnetCDF (4 Nodos)	102,60	9,43	± 2,62

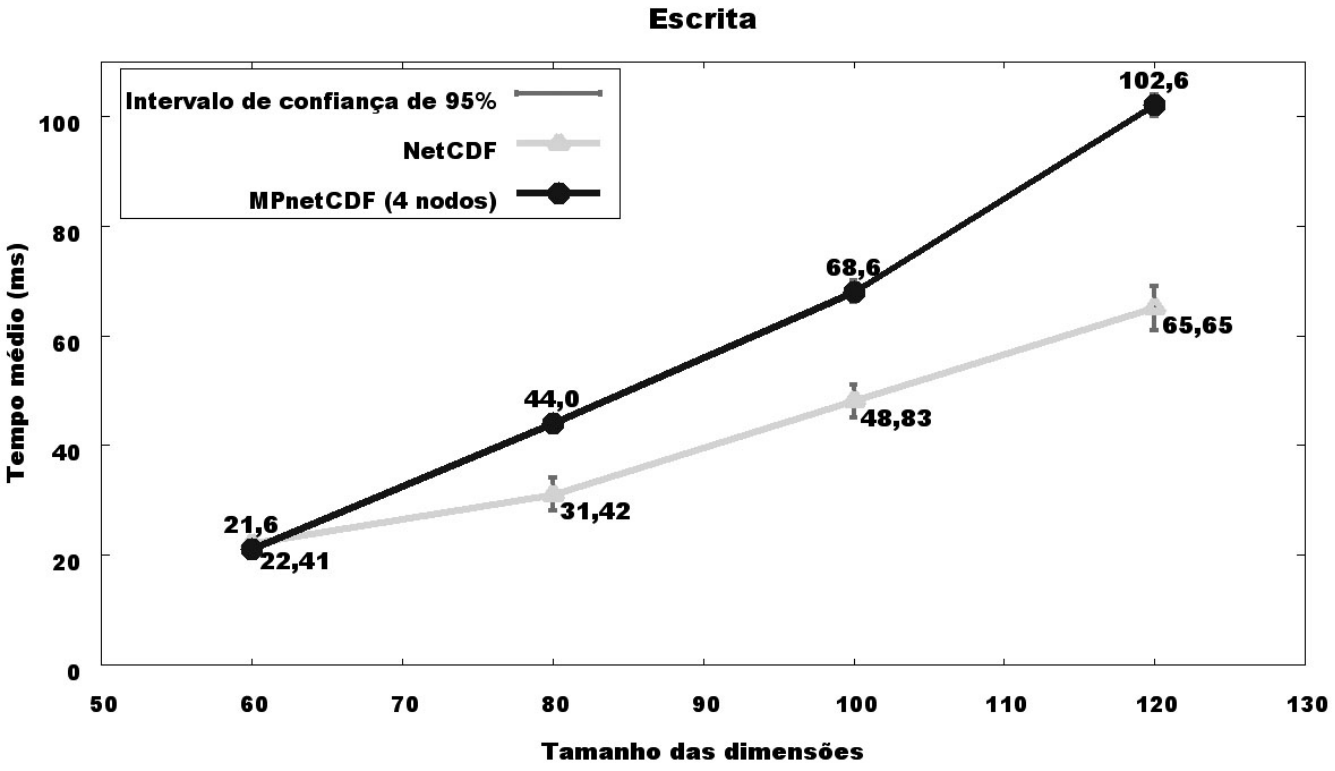

Figura 5.3: Gráfico com os tempos médios de escrita do NetCDF e do MPnetCDF (4 nodos).

abre o arquivo local, posiciona o braço de leitura no início do bloco e o lê por inteiro. Isso explica os tempos de leitura mais baixos obtidos para esse sistema em relação aos demais sistemas citados neste trabalho.

5.3 MPnetCDF Centralizado X NetCDF

Por fim, são mostradas as comparações entre o MPnetCDF executado de maneira centralizada, ou seja, com apenas um servidor instanciado no mesmo nodo da aplicação, e o NetCDF.

Também aqui, forão repetidos os valores obtidos para o NetCDF já mostrados na seção anterior, acresentando-se os referentes ao MPnetCDF centralizado, de forma a facilitar a comparação. A tabela 5.5 e a figura 5.5 contém os dados dos tempos de escrita.

Ao utilizar apenas um nodo no MPnetCDF, o tempo despendido para inicializar o sistema diminui significaticavamente. Também, os dados são escritos na memória volátil local, contribuindo na redução do tempo em relação a execução com vários servidores.

Na leitura, o netCDF é mais rápido. Em qualquer disco rígido, as operações de escrita diferem das de leitura, chegando a ser até quatro vezes mais lentas, o que seria suficiente para explicar a inversão no desempenho entre leitura e escrita que percebemos nessa comparação. Porém, ainda deve ser considerado o fato de que, mesmo estando no mesmo nodo, a aplicação e o servidor berkeley DB não compartilham o mesmo espaço de endereçamento na memória. Assim, ao realizar uma leitura a aplicação utiliza o *loopback* para acessar o servidor e a tabela de distribuição ainda é consultada, mesmo havendo só um servidor.A tabela 5.6 e a figura 5.6 mostram os resultados obtidos para as operações de leitura.

Tabela 5.4: Resultados dos tempos de leitura do NetCDF e do MPnetCDF (4 nodos).

x,y,z	Sistema	Tempo Médio (ms)	Desv. Padr.	IC
60	NetCDF	7,40	7,23	± 2,00
	MPnetCDF (4 Nodos)	34,80	6,46	± 1,79
80	NetCDF	14,20	5,75	± 1,59
	MPnetCDF (4 Nodos)	73,20	10,19	± 2,82
100	NetCDF	23,20	6,53	± 1,81
	MPnetCDF (4 Nodos)	113,80	9,87	± 2,74
120	NetCDF	32,40	9,60	± 2,66
	MPnetCDF (4 Nodos)	179,80	14,64	± 4,06

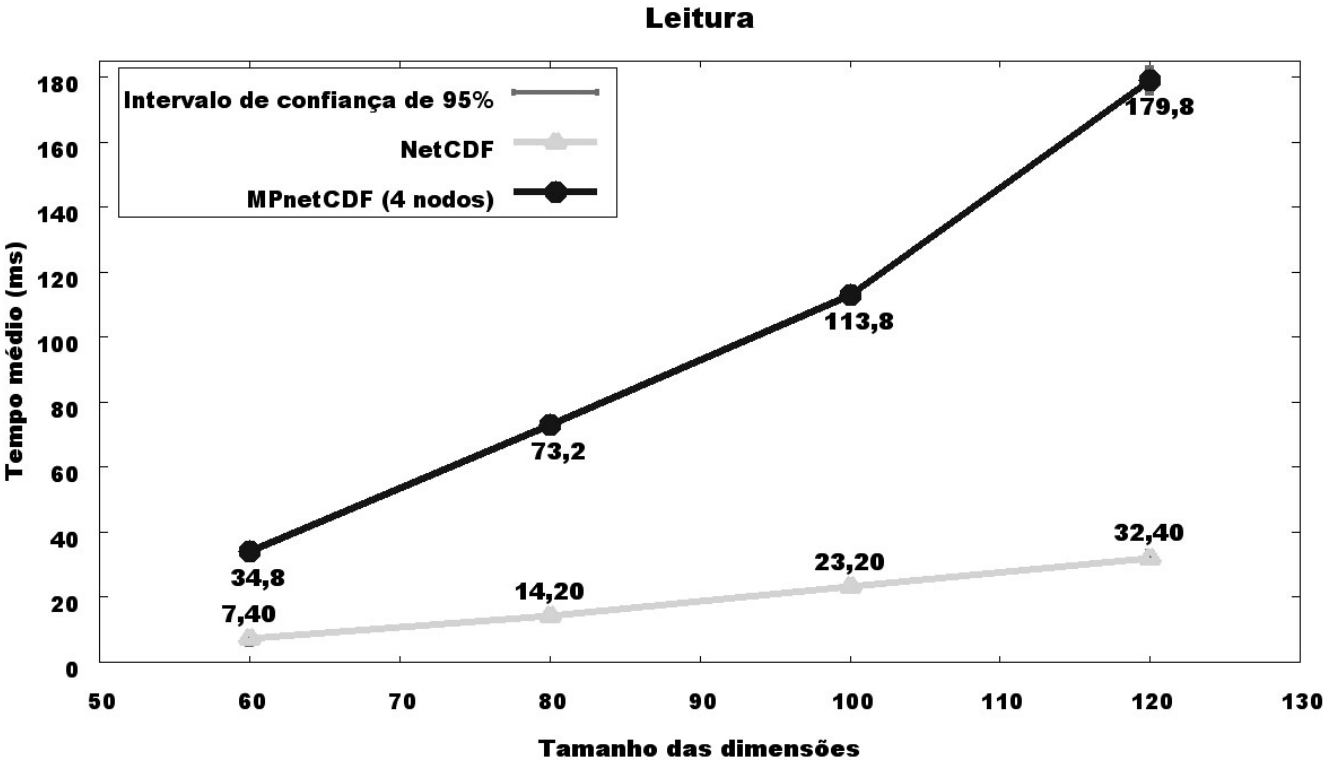

Figura 5.4: Gráfico com os tempos médios de leitura do NetCDF e do MPnetCDF (4 nodos).

Tabela 5.5: Resultados dos tempos de escrita do NetCDF e do MPnetCDF centralizado.

x,y,z	Sistema	Tempo Médio (ms)	Desv. Padr.	IC
60	NetCDF	22,41	5,91	± 1,64
	MPnetCDF Centr.	4,20	6,09	± 1,69
80	NetCDF	31,42	11,07	± 3,07
	MPnetCDF Centr.	24,20	7,02	± 1,95
100	NetCDF	48,83	13,80	± 3,83
	MPnetCDF Centr.	36,00	7,82	± 2,17
120	NetCDF	65,65	15,55	± 4,31
	MPnetCDF Centr.	45,60	12,32	± 3,41

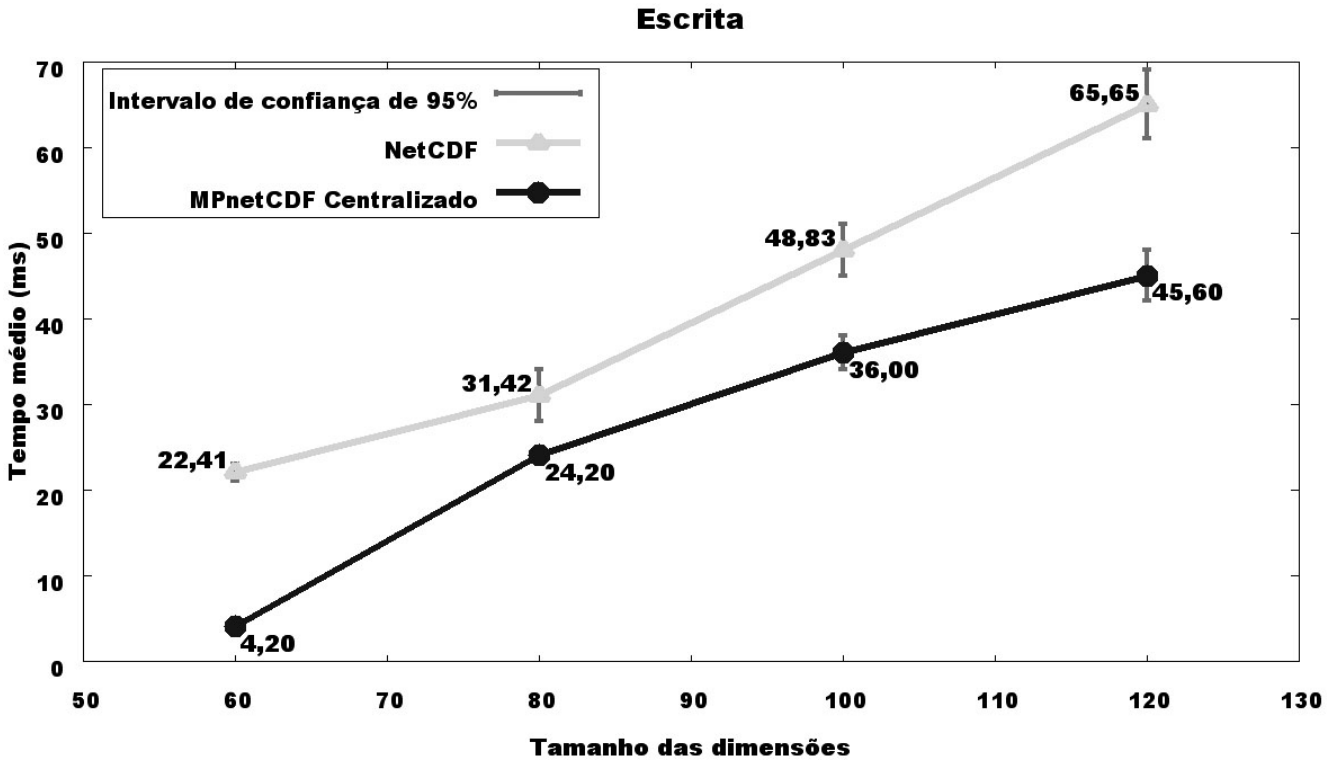

Figura 5.5: Gráfico com os tempos médios de escrita do NetCDF e do MPnetCDF centralizado.

Tabela 5.6: Resultados dos tempos de leitura do NetCDF e do MPnetCDF centralizado.

x,y,z	Sistema	Tempo Médio (ms) (ms)	Desv. Padr.	IC
60	NetCDF	7,40	7,23	± 2,00
	MPnetCDF Centr.	8,80	7,46	± 2,07
80	NetCDF	14,20	5,75	± 1,59
	MPnetCDF Centr.	19,20	6,34	± 1,76
100	NetCDF	23,20	6,53	± 1,81
	MPnetCDF Centr.	36,20	6,97	± 1,93
120	NetCDF	32,40	9,60	± 2,66
	MPnetCDF Centr.	62,40	7,97	± 2,21

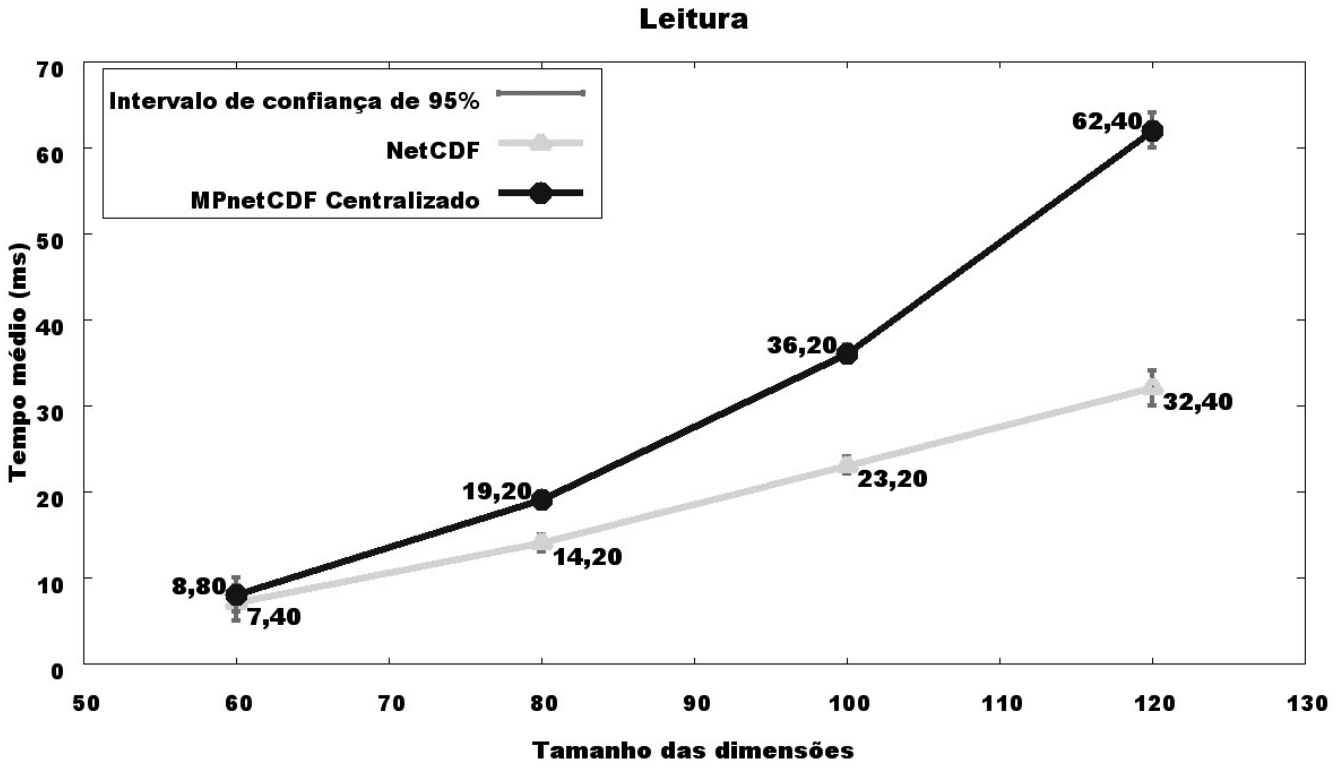

Figura 5.6: Gráfico com os tempos médios de leitura do NetCDF e do MPnetCDF centralizado.

Capítulo 6

Conclusão

Neste trabalho foi proposta uma solução de distribuição dos dados de uma aplicação sobre uma rede de computadores, especialmente um cluster, utilizando um sistema de gerenciamento de bancos de dados em memória - SGBDM, com acesso remoto via tecnologia cliente-servidor. Para tal, uma instância do servidor deve ser instalada em cada nodo e sobre eles uma camada de convergência capaz de gerenciar a distribuição dos dados, de forma adaptada a aplicação.

A idéia foi apresentada através da descrição de um caso exemplo que utiliza como camada de armazenamento o Berkeley DataBase e como aplicação o Network Common Data Form. O sistema foi testado e validado. Testes foram descritos e apontaram as vantagens e potencialidades do método.

A proposta mostrou-se adequada, tanto no que diz respeito aos aspectos gerais que sugeriram a utilização de camada de convergência que adapte um SGBDM a uma aplicação, como particularmente proporcionou a obtenção de uma versão distribuída de um importante sistema muito utilizado em aplicações de geociências, o netCDF.

Os resultados mostraram tempos de leitura e escrita menores que os de outras versões distribuídas, embora, como era esperado, maiores em relação a versão centralizada. Futuramente, medidas referentes a paralelização de tarefas de comunicação com os servidores devem ser adotadas, reduzindo ainda mais os tempos.

Ainda, foram testadas com sucesso os acessos diretos a elementos localizados no meio da matriz e, efetivamente, podem ser inseridos ou removidos dados no cabeçalho sem que seja necessária a movimentação do dataset entre regiões de memória de tamanho

compatível.

Como deficiência da proposta, cita-se a restrição de que os dados distribuídos devem ser acessados de forma bem comportada, por exemplo, por aplicações centralizadas, ou por aplicações distribuídas que não produzam deadlocks distribuídos.

Trabalhos futuros para resolver essa restrição devem ser desenvolvidos e envolvem a utilização de gerenciamento de transações distribuídas. Ainda, deve ser perseguida uma solução para possibilitar a criação de variáveis com múltiplas dimensões ilimitadas.

Também, deverá ser desenvolvida uma aplicação que transforme uma base de dados já criada e preenchida do modelo clássico em uma banco de dados do Berkeley DB, para que os dados inseridos não sejam perdidos ou precisem ser reinseridos no MPnetCDF.

Bibliografia

[Dickens and Thakur, 1998] Dickens, P. M. and Thakur, R. (1998). A performance study of two-phase i/o. *Euro-Par, 1998.*

[Filho and Pedone, 2008] Filho, N. D. and Pedone, F. (2008). Dsmdb: A distributed shared memory approach for building replicated database systems. *WDDDM'2008.*

[HDF-Group, 2008] HDF-Group (2008). HDF5 Tutorial. `http://www.hdfgroup.org/HDF5/Tutor/` (Acessado em: 13/mar/2009).

[HDF-Group, 2009a] HDF-Group (2009a). About The HDF Group. `http://www.hdfgroup.org/about/` (Acessado em: 28/abr/2009).

[HDF-Group, 2009b] HDF-Group (2009b). HDF4 Documentation. `http://www.hdfgroup.org/products/hdf4/index.html` (Acessado em: 13/mar/2009).

[IBM, 2004] IBM (2004). IBM's Watson-Lab. MPI I/O. `http://beige.ucs.indiana.edu/I590/node86.html` (Acessado em: 28/abr/2009).

[Li et al., 2003] Li, J., keng Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. (2003). Parallel netcdf: A high-performance scientific i/o interface. `http://trac.mcs.anl.gov/projects/parallel-netcdf` (Acessado em: 26/jul/2009).

[Nitzberg and Lo, 1991] Nitzberg, B. and Lo, V. (1991). Distributed shared memory: A survey of issues and algorithms. *IEEE Computer*, 24(8):52-60.

[Open-MPI, 2009] Open-MPI (2009). Open MPI Frequently Asked Questions. `http://www.open-mpi.org/faq/` (Acessado em: 22/jul/2009).

[Qureshi and Rashid, 2005] Qureshi, K. and Rashid, H. (2005). A performance evaluation of rpc, java rmi, mpi and pvm. *Malaysian Journal of Computer Science*, 18:38-44(2).

[Rew et al., 2008a] Rew, R., Davis, G., Emmerson, S., Davies, H., and Hartnett, E. (2008a). The NetCDF C Interface Guide. NetCDF Version 4.0. Last Updated 27 June 2008.

[Rew et al., 2008b] Rew, R., Davis, G., Emmerson, S., Davies, H., and Hartnett, E. (2008b). The NetCDF Users Guide. Unidata Program Center. Data model, programming interfaces and format for self-describing portable data of NetCDF Version 4.0.

[Sirinavasan, 1995] Sirinavasan, R. (1995). Rfc1832 - xdr: External data representation standard. *Request for Comments: 1832, Category: Standards Track.*

[Sleepycar-Group, 2009] Sleepycar-Group (2009). Oracle Berkeley Database Documentation. `http://www.oracle.com/technology/documentation/index.html` (Acessado em 13/mar/2009).

[Srinivasan, 1995] Srinivasan, R. (1995). Rpc: Remote procedure call protocol specification version 2. *RFC 1831.* `http://www.rfc-archive.org/getrfc.php?rfc=1831` (Acessado em: 13/mar/2009).

[SSH, 2009] SSH (2009). SSH User Documentation. `http://www.ssh.com/support/documentation/` (Acessado em: 22/jul/2009).

[Tanenbaum, 2002] Tanenbaum, A. S. (2002). *Modern Operating Systems.* Prentice Hall, 2 edition.

[Tenenbaum et al., 1995] Tenenbaum, A. A., Langsam, Y., and Augenstein, M. J. (1995). *Estruturas de Dados Usando C.* Makron Books, 1 edition.

[Thakur et al., 1999] Thakur, R., Gropp, W., and Lusk, E. (1999). Parallel netcdf: A high-performance scientific i/o interface. *Frontiers of Massively Parallel Computation, 1999.*

[Unidata, 2009] Unidata (2009). Unidata Overview. `http://www.unidata.ucar.edu/publications/directorspage/UnidataOverview.html` (Acessado em: 26/abr/2009).

[UT, 1995] UT (1995). University of Tennessee. MPI: A Message- Passing Interface Standard. http://www.mcs.anl.gov/research/projects/mpi/ (Acessado em: 28/abr/2009).

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo