UM *FRAMEWORK* PARA MEMÓRIA COMPARTILHADA DISTRIBUÍDA E ESTUDO DE CASO EM APLICAÇÕES DE GEOCIÊNCIAS SOBRE *CLUSTERS* DE COMPUTADORES

JEISON VIEIRA[1], SILVIA COSTA BOTELHO[2], NELSON DUARTE FILHO[3]

RESUMO – Este artigo descreve como distribuir dados das aplicações usando um *middleware* constituído por um Sistema de Gerenciamento de Banco de Dados em Memória como camada base; e uma camada de convergência adequada as aplicações. Um servidor do SGBDM deve ser instalado em cada nodo de uma rede de computadores e sobre ele uma camada de convergência que distribui de modo transparente os dados das aplicações. As idéias são apresentadas através da implementação de um caso exemplo usando o Berkeley DataBase como camada de armazenamento e uma camada de convergência para o Network Common Data Form – NetCDF, padrão usado na área de geociências para armazenar grandes volumes de dados.

A FRAMEWORK FOR DISTRIBUTED SHARED MEMORY - CASE STUDY IN GEOSCIENCE APPLICATIONS ON CLUSTERS OF COMPUTERS

ABSTRACT — This paper describes how to distribute application data using a middleware composed of a remote access Memory Management Data Base as base layer and a convergence layer to the application. One server of de MMDB must be installed on each node of a computer network and over it a convergence layer that distributes the data transparently. The ideas are presented through a case example implementation using Berkeley DataBase as storage layer and a convergence layer to Network Common Data Form – NetCDF, standard used in geosciences to store large volumes of data.

1 FURG; MsC; jeison.ecomp@pop.com.br
2 FURG; Dr; silviacb@furg.br
3 FURG; Dr; dmtnldf@furg.br

1. INTRODUÇÃO

O aumento da capacidade das memórias RAM dos computadores e a diminuição dos seus custos, aliados a possibilidade de interligar computadores através de redes com altas taxas de transmissão de dados, também a baixo custo, gera um cenário favorável às aplicações, 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 de computadores (*cluster*). Isso traz vantagens óbvias de desempenho, em função dos baixos tempos de acesso às memórias RAM, quando comparados aos dos meios de armazenamento magnéticos, como os discos rígidos. Entretanto, sob esse ponto de vista, essas arquiteturas ainda costumam ficar subutilizadas, tanto devido à falta de cultura para elaboração de *software* paralelo e distribuído, quanto a falta de soluções de ferramentas automatizadas disponíveis para tal. Outros trabalhos, ver [6], vêm sendo desenvolvidos, buscando obter maior vantagem desse tipo de ambiente, fazendo uso, por exemplo, de DSM (*Distributed Shared Memory*) [10] como solução para compartilhamento e distribuição dos dados. Em especial, citamos sistemas de gerenciamento de espaços de armazenamento distribuídos, que visam oferecer um espaço único de endereços às aplicações. Essas propostas costumam 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 memória com semântica de consistência que admita aplicações concorrentes genéricas. Por exemplo, consistência estrita.

Neste artigo, propõe-se uma abordagem de gerenciamento de dados distribuídos, através de uma variação do modelo DSM, objetivando 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 *framework* proposto se constitui de:

- Uma camada de serviço, implementada sobre SGBDs centralizados, executando paralelamente;

- Uma camada de distribuição capaz de manter uma consistência de memória adequada às características da aplicação; e
- A camada de aplicação propriamente dita.

A proposta foi implementada utilizando o *Berkeley DataBase* – BDB [3], executando sobre os nodos de um *cluster.* Como o BDB é um banco de dados centralizado, capaz de gerenciar tabelas totalmente em memória, se pode avaliar a distribuição proposta, que objetiva disponibilizar mais memória RAM e mais poder de processamento à aplicação, cujo código executa de forma centralizada. A camada de distribuição oferece um modelo de consistência adequado às necessidades da aplicação, mais restrito do que os sistemas de gerenciamento do tipo DSM, mas possibilitando a utilização de várias instâncias de um sistema de banco de dados centralizado executando paralelamente. O BDB foi utilizado por ser livre, aberto e referência de desempenho entre sistemas de gerenciamento de bancos de dados em memória.

A idéia de utilizar esse *framework* surgiu da percepção de que uma especificação muito utilizada nas geociências, o *Network Commom Data Form* – NetCDF [5], para armazenamento de grandes volumes de dados, originalmente implementada de forma centralizada, pode ser tornada distribuída, sem que os seus usuários necessitem mudar o paradigma de programação sequencial, permitindo que até mesmo programas anteriores que usam o padrão possam tirar proveito da distribuição dos dados aqui descrita, de modo transparente.

A figura 1 apresenta uma representação esquemática da estrutura implementada, com três nodos compondo o *cluster*, cada um com um servidor BDB instalado, constituindo a camada de serviço. A API da camada de distribuição é semelhante a do BDB, tendo uma camada de convergência adaptada a ela. Ou seja, as chamadas do NetCDF ao sistema operacional foram substituídas por chamadas à API do conjunto de servidores. Com isso se obteve uma solução NetCDF distribuída, com a distribuição dos dados tornada transparente.

Na esteira de adaptar o NetCDF ao sistema implementado, denominado MpnetCDF, foram também realizadas melhorias na própria aplicação, que serão comentadas ao longo do artigo.

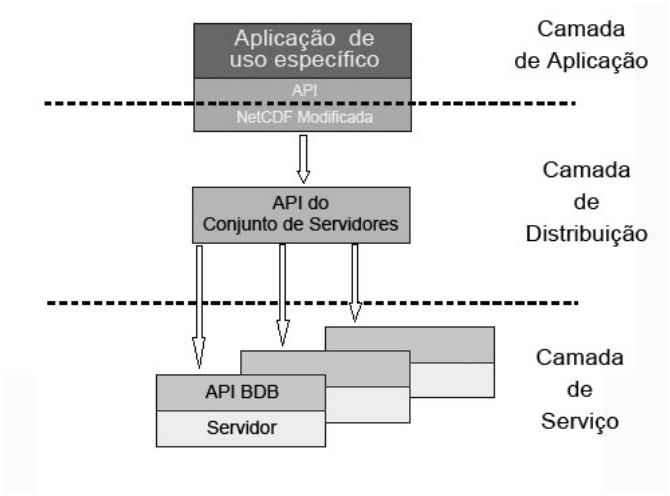

Figura 1. Representação em camadas do MPnetCDF

Na próxima seção se descreve o BDB e na seção III o NetCDF, ambos de forma simplificada, com vistas a apenas introduzir os conceitos básicos necessários a melhor compreensão da proposta apresentada. Na seção IV é descrito o MPnetCDF, conforme ele foi implementado, e na seção V apresentam-se alguns resultados com ele obtidos. Finalmente, a seção VI conclui o trabalho, fornecendo algumas sugestões de desenvolvimentos futuros.

2. O BERKELEY DATABASE

O BDB [3] 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.

A. Utilização Local

Para utilizar o BDB localmente há que 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 subespaços de endereços paginados, sendo que cada subespaço pode conter um ou mais bancos de dados. Na figura 2 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. Ali, 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 multiprocessos, ele garante a manutenção de consistência dos dados envolvidos.

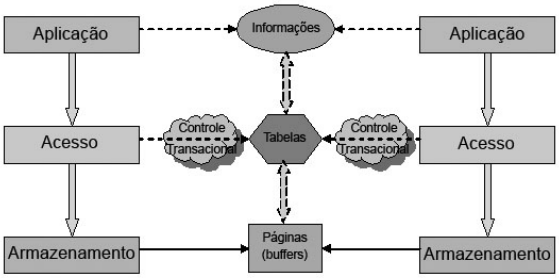

Figura 2. Esboço da organização de uma sessão de uso local do BDB.

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 em páginas são inseridas e recuperadas dos subespaços de armazenamento através do acionamento de métodos da camada de ARMAZENAMENTO.

B. 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; 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 nestas executados invocam o servidor, via RPC, para que este acione os métodos locais a eles correspondentes. Na figura 3, 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. 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 aplicações.

3. O NETCDF

O NetCDF é uma especificação proposta pela Unidata [1], geralmente implementada sob a forma de uma biblioteca de funções de acesso a dados armazenados na forma de matrizes. Um matriz, é uma estrutura retangular *n*-dimensional (onde *n* é 0, 1, 2, ...) contendo itens do mesmo tipo (ex.: caracteres de 8 bits, inteiros 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 portáveis 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, por exemplo, as unidades de medida adotadas para os dados. “Portáveis” significa que os dados em um *dataset* são representados em uma forma que pode ser acessada por computadores de diferentes arquiteturas. Os valores nas matrizes podem ser acessados diretamente, sem 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.

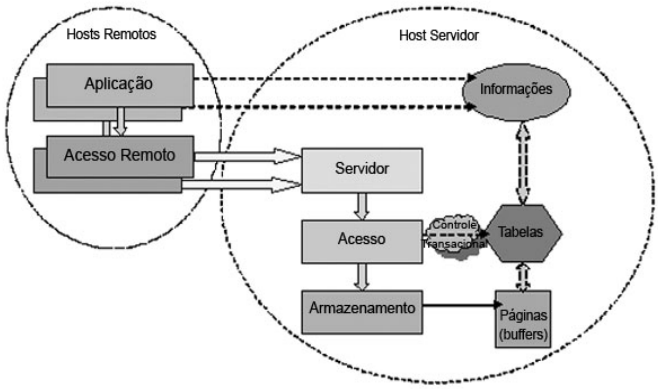

Figura 3. Ilustração do acionamento remoto do BDB por duas aplicações.

A. Os Formatos de Arquivos NetCDF

Até a versão anterior a 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 demais 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. Os formatos clássico e 64-bits são freqüentemente referidos na documentação oficial como NetCDF-3. Na versão 4.0.0 um novo formato binário surgiu, o formato HDF5, que trouxe consigo a eliminação de algumas restrições existentes nos formatos anteriores. Esse modelo é referido na documentação oficial como NetCDF-4.

B. O Modelo Descritor de Dados do NetCDF

Os *datasets* contem três informações principais para descrever os dados: dimensões, variáveis e atributos [5], todas possuindo um nome e um número de identificação pelos quais podem ser referenciados. Esses componentes são usados para dar significado aos dados e as relações entre os campos das matrizes existentes nos *datasets*. A dimensão é geralmente usada para representar uma característica física, como tempo, latitude, altura, etc. É descrita por um nome e um tamanho, que indica o número de elementos que a compõe. Se ela for do tipo *unlimited* poderá ter novos elementos adicionados, até o limite de armazenamento do *dataset* e será chamada *record* variable, aqui chamada de dimensão ilimitada. No NetCDF-3 só pode haver uma dimensão ilimitada. 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. Ela, a variável, tem um nome, um tipo e uma forma, descrita pela sua lista de dimensões, informada no momento da sua criação. Uma variável também pode ter atributos associados. Eles são usados para armazenar dados sobre os dados (metadados), como a unidade de medida adotada ou o intervalo de valores válidos. 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 *record variable* também é obtida do mesmo modo, mas nesse caso o produto é variável porque envolve a dimensão ilimitada, a qual pode variar. O tamanho da dimensão ilimitada é igual ao número de registros. Um registro é o conjunto de dados obtidos a partir das dimensões não ilimitadas para cada variação unitária do índice da dimensão ilimitada. Por exemplo, imaginemos uma variável de registros que armazene valores de pressões em função das coordenadas latitude, longitude e tempo, sendo este último a dimensão ilimitada. Para cada instante em que uma medição for feita será armazenado um registro contendo três valores: latitude, longitude e pressão. Se forem feitas dez medições teremos dez registros. 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.

C. Estrutura dos Arquivos

Não é necessário conhecer a estrutura dos arquivos para realizar leituras e escritas de dados através do NetCDF porque tais operações ficam escondidas sob a interface oferecida pelas funções. Entretanto, esse conhecimento auxilia no entendimento das limitações apresentadas pelo NetCDF e as melhorias introduzidas na proposta aqui apresentada, conforme mostrado na próxima seção. 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. O cabeçalho, no início do arquivo, contém informações sobre as dimensões, 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 relativos das variáveis 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 necessário a completar o espaço último bloco no disco. Isso traz como vantagem arquivos NetCDF compactos, necessitando um *overhead* mínimo para armazenar os metadados que tornam os *datasets* auto-descritivos. Entretanto, uma desvantagem disso é que qualquer operação que resulte no aumento ou diminuição dos dados do cabeçalho, como, por exemplo, adicionar novas dimensões ou variáveis, requer que os dados sejam copiados para uma outra área de dados compatível com o novo tamanho. O mesmo problema ocorre ao criar uma nova matriz de tamanho fixo. Como alternativa, é possível especificar explicitamente, no momento da criação do *dataset*, um tamanho extra a ser reservado para o caso de futuras alterações. Quando o tamanho do cabeçalho muda, dados no arquivo são movimentados e os seus endereços mudam. Se outro programa estiver lendo o NetCDF durante a redefinição, a sua visão dos dados será baseada nos índices antigos, possivelmente incorretos. Se houver necessidade de uma redefinição enquanto o *dataset* estiver sendo compartilhado, então um mecanismo externo à biblioteca NetCDF deverá ser empregado, para prevenir o acesso ao *dataset* durante a redefinição. Na versão aqui adaptada, além de não haver necessidade de cópias no caso de alterações, a consistência dos dados pode ser mantida pelo BDB. A área de dados fica subdividida em duas partes. A primeira corresponde aos dados de tamanho fixo e 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 contendo dados referentes as variáveis do tipo *record variable*. Nestas, os dados são armazenados contiguamente por registro e não por variável; os registros das diversas variáveis ficam intercalados. A figura 4 apresenta uma representação da estrutura descrita.

4. O MPNETCDF

O MPnetCDF é a mais importante materialização das idéias propostas neste trabalho. E, nesse sentido, o seu principal componente é a camada intermediária de *software* capaz de distribuir os dados da aplicação – o NetCDF – entre os servidores que constituem o gerenciador de armazenamento, composto por instâncias de um sistema de gerenciamento de bancos de dados em memória – o BDB - instalados nos nodos de uma rede de computadores – um *cluster*. Além dos motivos acima apresentados, o BDB foi escolhido como camada base do MPnetCDF por suportar objetos multidimensionais (matrizes) como unidades básicas de acesso aos dados. Ele gerencia tabelas compostas por dois campos: *chave* e *valor*, que, dependendo do método de acesso utilizado, podem receber estruturas de dados complexas. Porções do campo *valor* podem ser acessados sem que seja necessário copiar toda a estrutura que o constitui para a área de memória da aplicação, o que possibilita eficiência na indexação de uma matriz que esteja contida no campo *valor*, por exemplo. Afora isso, como o BDB oferece suporte a acesso remoto, ele é adequado para possibilitar a distribuição dos dados.

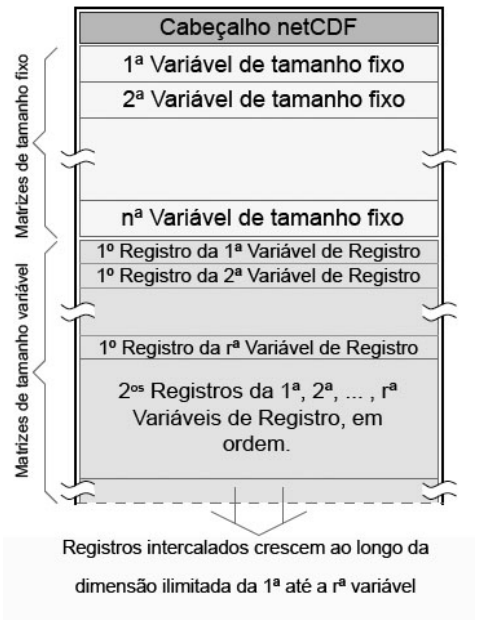

Figura 4. *Dataset*, dividido em cabeçalho e área de dados.

A seguir, apresentam-se as modificações na forma de armazenamento e na

estruturação dos dados resultantes das idéias propostas.

A. Distribuição dos Dados

No BDB vários bancos de dados podem ser encapsulados em um *environment*, ambiente utilizado para possibilitar acesso concorrente entre vários processos ou *threads*. Supõe-se que as aplicações que estiverem 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. Cada servidor mantêm um *environment*, que reúne seu conjunto de bancos de dados. Por decisão de projeto, cada banco de dados é fisicamente armazenado em um arquivo. Ao fazer a adaptação dos sistemas, foi determinado que cada *dataset* (arquivo NetCDF) ficaria armazenado em um banco de dados (arquivo BDB). 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, 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 5. 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 à API original. Uma delas possibilita indicar o caminho e o nome do arquivo que contém a lista de endereços IP dos nodos onde há servidores BDB instalados. Por padrão, os *datasets* são criados de forma circular. Ou seja, havendo dois servidores, a primeira solicitação de criação será encaminhada ao primeiro da lista de IPs, a segunda para o segundo, a terceira novamente para o primeiro, e

assim sucessivamente. A outra função serve para oferecer à aplicação a possibilidade de determinar em quais servidores os arquivos devem ser criados. Isso transfere à aplicação o arbítrio de determinar qual a distribuição dos dados é a mais favorável.

B. Estrutura dos Dados

Diferentemente das implementações do NetCDF geralmente encontradas, que conforme mostrado na figura 4 armazenam os dados de forma sequencial, no MPnetCDF pode ser utilizado o método de acesso B-tree [2]. Essa estrutura de armazenamento implica o relaxamento da restrição de que para qualquer modificação do *dataset* que leve ao aumento ou diminuição do cabeçalho ou da área de dados de tamanho fixo torna-se necessário a cópia dos dados do arquivo para uma área de armazenamento compatível com o novo tamanho. Ou seja, na solução aqui apresentada as chaves dos pares 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.

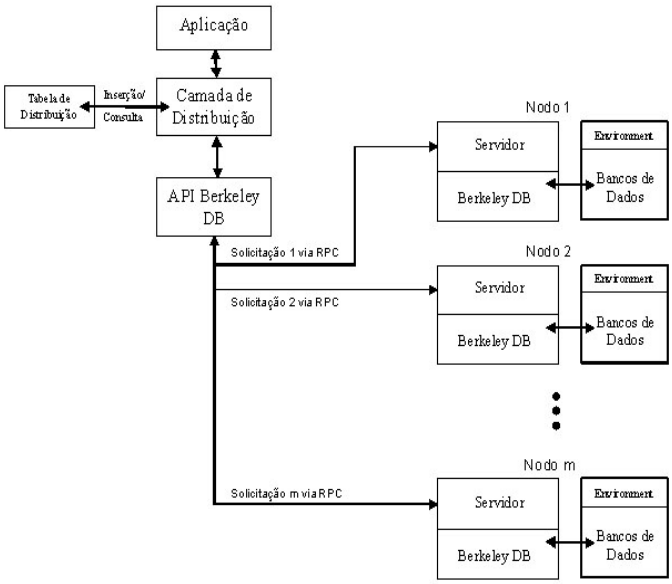

Figura 5. Funcionamento esquemático do MpnetCDF.

Na figura 6 é possível observar que, diferentemente do modo de armazenamento contíguo intercalado utilizado para as variáveis de registro no NetCDF clássico, cada variável aponta para seu próprio conjunto de registros e elas são independentes entre si. No caso das variáveis comuns elas apontam simplesmente para o conjunto de informações ao qual se referem. 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*.

5. TESTES E RESULTADOS

A avaliação do MPnetCDF foi realizada num *cluster* constituído por dez computadores: um *front-end*, um *master* e oito *slaves*. Os testes foram realizados 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*. Para obtenção dos resultados foi utilizado um *benchmark* que acompanha o pacote PnetCDF, escrito para executar sobre quatro processadores. O PnetCDF é uma versão distribuída do NetCDF e foi usada como referência por ser, até o momento, a versão distribuída de melhor desempenho do NetCDF. Desenvolveu-se então uma versão similar a esse *benchmark*, 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. O teste foi escolhido por utilizar todos os tipos de variáveis oferecidas pelo NetCDF, oportunizando a comparação 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.

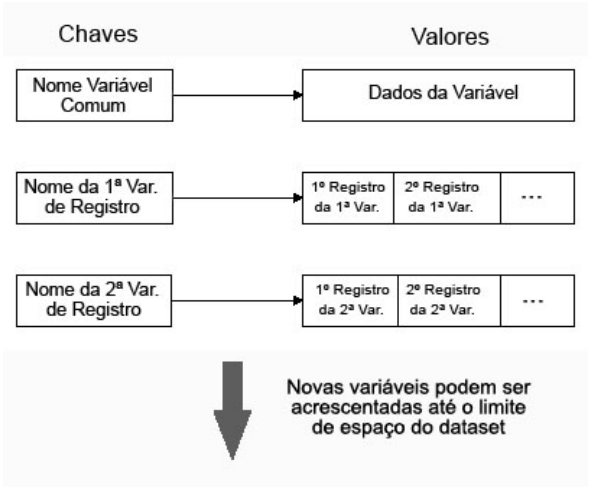

Figura 6. Estrutura das variáveis no MpnetCDF.

Para avaliar 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. Os resultados foram então cotejados com os obtidos pelo PnetCDF. 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 de 95% e o desvio padrão das amostras, tanto para operações de escrita como de leitura. 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, acredita-se que os resultados não seriam diferentes aos encontrados.

Na figura 7 se pode observar tempos médios sensivelmente menores no 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 [9], apresenta maiores latências e menores larguras de banda de passagem, quando comparado ao RPC (paradigma sobre o qual está implementada a comunicação em rede do MpnetCDF).

Em condições de testes rigorosamente idênticas, os desvios padrão em torno

da média também são sensivelmente menores, mostrando que o MPnetCDF tem um comportamento mais estável. 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. Nos gráficos, as barras verticais representam o intervalo de confiança do tempo médio encontrado, com um grau de certeza de noventa e cinco por cento. Devido a escala, as barras referentes ao MPnetCDF não são visualizadas.

Os resultados mostrados na figura 8 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 à medida que o tamanho das matrizes aumenta.

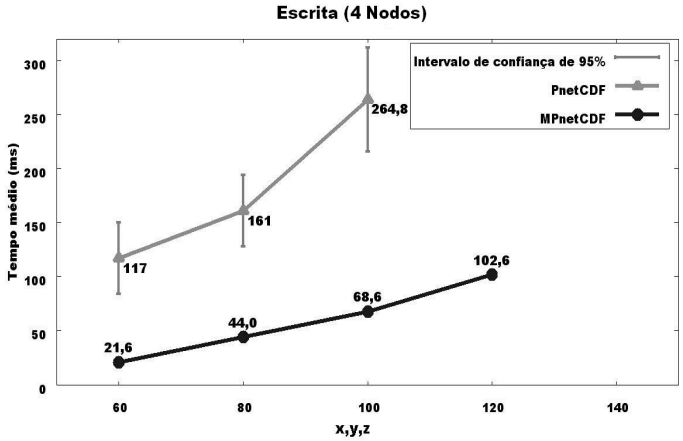

Figura 7. Comparação entre MPnetCDF e PnetCDF para escrita.

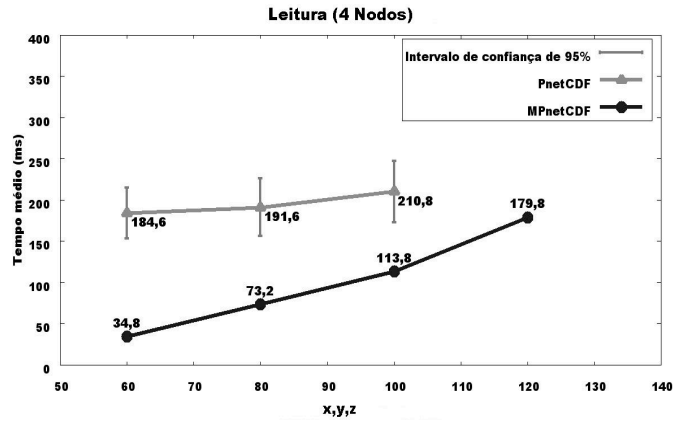

Figura 8. Comparação entre MPnetCDF e PnetCDF para leituras.

6. CONCLUSÃO

Neste trabalho foi apresentada uma solução de distribuição dos dados de uma aplicação sobre uma rede de computadores, especialmente um *cluster*, através de uma variação do modelo *DSM,* com manutenção de consistência relaxada, adequada à aplicação. Para isso, se propôs a utilização de um sistema de gerenciamento de bancos de dados em memória - SGBDM. Uma instância do SGBDM deve ser instalada em cada nodo do *cluster* 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*.

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 importante utilitário muito usado em aplicações das geociências. O utilitário foi também melhorado em relação à sua implementação padrão.

Como deficiência do modelo, 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. Particularmente, como o BDB pode ser usado como um gerenciador de recursos XA-compatível [7] e é sabido que implementações com essa característica podem operar em conjunto com o gerenciador de transações Tuxedo [8], essa é uma solução que pode ser perseguida.

REFERÊNCIAS

[1] http://www.unidata.ucar.edu/publications/directorspage/UnidataOverview.html (Acessado em 29/03/2009).

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

[3] Sleepycat-Group (2009). "*Oracle Berkeley Database Documentation*." http://www.oracle.com/technology/documentation/index.html (Acessado em 13/mar/2009).

[4] 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.

[5] 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.*

[6] Duarte Filho, N., Pedone, F., "*DSMDB: A Distributed Shared Memory Approach for Building Replicated Database Systems*", WDDDM'2008, Glasgow, Scotland, Março de 2008.

[7] "*X/Open CAE Specification Distributed Transaction Processing: The XA Specification"*, X/Open Document Number: XO/CAE/91/300.

[8] http://www.oracle.com/products/middleware/tuxedo/tuxedo.html.

[9] 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).

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