CLAUDIA DE ANDRADE TOCANTINS

ARMAZENAMENTO E MANIPULAÇÃO DE MAPAS VETORIAIS EM UMA ARQUITETURA PARALELA

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA

Rio de Janeiro, 30 de agosto de 1995

Claudia de Andrade Tocantins

ARMAZENAMENTO E MANIPULAÇÃO DE MAPAS VETORIAIS EM UMA ARQUITETURA PARALELA

Dissertação apresentada ao Departamento de Informática da PUC-RJ como parte dos requisitos para a obtenção do título de Mestre em Ciências em Informática.
Orientador: Marco Antônio Casanova

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 30 de agosto de 1995 (Data da defesa)

Aos sonhos, ao mar e à lua. Companheiros de sempre.

AGRADECIMENTOS

É impossível resumir em tão poucas palavras o carinho que sinto por todos aqueles que de alguma forma me incentivaram durante o meu mestrado.

Gostaria de agradecer em especial a Marco Antonio Casanova, meu orientador, pela convivência, críticas e sugestões que permitiram a realização deste trabalho.

Agradeço também:

À PUC-Rio que me fez *Engenheira* e agora *Mestre,* responsável por boa parte de minha formação.

À IBM Brasil, em especial, ao Centro Científico Rio, onde este trabalho foi desenvolvido.

À Capes, pelo suporte financeiro.

Ao INPE e às demais instituições envolvidas no projeto GEOTEC do Programa Temático em Ciência da Computação do CNPq, pela introdução aos problemas da área de geoprocessamento.

Mais importante que as instituições, são as *pessoas* que as formam. Assim, não posso deixar de lembrar de todos que conviveram comigo nesse tempo. Gostaria de agradecer:

Aos meus pais e irmãos, familiares e amigos, pelo amor e confiança.

Às minhas afilhadas, Claudia e Carla e às amigas, Lilian e Cristiana, pela paciência e pela amizade, próximas e distantes.

A Luiz Tucherman que acreditou em mim e me proporcionou a convivência enriquecedora de tantos na IBM.

Aos amigos que fiz no Centro Científico Rio (até mesmo aos mais implicantes) e na IBM em geral.

Aos professores da PUC e também a meus colegas na Informática, pelas aulas, discussões, churrascos, chopps, festas... Em especial ao pessoal que frequenta o chopp das quartas-feiras que não me deixou esquecer que antes de qualquer coisa, eu era *aluna da PUC.*

Quero, ainda, prestar uma homenagem a Eduardo Amarante, uma pessoa que tem feito falta e que certamente gostaria de estar vendo a concretização deste trabalho.

RESUMO

Como forma de melhorar o desempenho de operações em bancos de dados geográficos, essa dissertação propõe métodos de armazenamento e manipulação de mapas vetoriais num ambiente paralelo.

O trabalho resume, inicialmente, um modelo para objetos geográficos, incluindo a definição de conceitos ligados a mapas vetoriais. Define também uma arquitetura genérica para gerenciadores de objetos geográficos e discute variações desta arquitetura para um ambiente distribuído.

Em seguida, o texto aborda em mais detalhes a estrutura de armazenamento proposta e suas variações. Descreve rotinas de edição da estrutura e heurísticas para a distribuição uniforme do banco de dados. Finalmente, a dissertação define algumas operações geográficas e o procedimento para resolve-las de forma paralela, visando minimizar o custo de sua execução.

ABSTRACT

Motivated by the need to improve the performance of the execution of spatial operations in a geographical database, this dissertation proposes a method to store and manipulate vector maps in a parallel environment.

First, a model for geographic objects and some concepts about vector maps are summarized. A general architecture for geographic objects management is then presented and some variations for a distributed environments are discussed.

Then, the storage structure and its variations are explained in detail. Procedures to edit the structure and some heuristics to define an uniform load balancing are described. Finally, some geographic operations are defined and their parallel execution is discussed.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Sistemas de Geoprocessamento, também chamados de Sistemas de Informação Geográfica, ou SIGs caracterizam-se por modelar e analisar *dados georeferenciados*, ou seja, dados que descrevem objetos e fenômenos do mundo real associados a uma localização na superfície terrestre [Goo91]. Além do aspecto de georeferenciamento, os dados utilizados em SIGs caracterizam-se pelo grande volume, pela heterogeneidade e pelo fato de estarem potencialmente distribuídos em diversos locais. Uma série de relacionamentos complexos podem ser definidos entre eles, como por exemplo, adjacência e cruzamento da geometria de representações espaciais.

As operações típicas de SIGs, designadas no que se segue simplesmente por *operações geográficas*, são também muito mais complexas do que operações convencionais, pois em geral tratam de relacionamentos espaciais entre representações de objetos. Os requisitos de um banco de dados geográfico são especialmente relacionados à eficiência do processamento de consultas. Portanto, do ponto de vista de gerência e manipulação de dados, SIGs colocam desafios interessantes.

Diversos estudos têm sido desenvolvidos com o intuito de melhorar o desempenho dos SIGs. Eles abrangem áreas tais como a definição de arquiteturas e modelos de dados [WHM90, Goo92, Her92, RM92, FK93] e o uso de novos métodos de acesso espacial [Sam90, BKSS90, Cox91, HS92, MCD94]. Essas estruturas tendem a aumentar a capacidade de armazenamento de um banco de dados geográfico, permitindo que largas extensões de dados sejam manipuladas em conjunto. Nem sempre, porém, o filtro exercido por estes métodos é suficiente para desconsiderar a complexidade das operações geográficas. A síntese de algoritmos eficientes para a resolução de consultas espaciais é assunto por exemplo de [BKSS94, HS94, Wan93].

Com o advento do uso de arquiteturas paralelas é razoável que se estude seu uso para incrementar o desempenho dos bancos de dados geográficos. Como vantagens diretas do uso de um ambiente distribuído, obtém-se uma maior capacidade de armazenamento sem que haja perda de desempenho, já que múltiplos processadores cooperam para a resolução das operações. Novos problemas, porém, surgem como, por exemplo, a complexidade na administração dos dados; considerações sobre sua distribuição; problemas de balanceamento de carga, etc...

Este trabalho alinha-se com este esforço endereçando o problema de armazenamento distribuído de um grande volume de dados georeferenciados, organizados sob forma de mapas, de forma a paralelizar a execução de operadores espaciais e reduzir o custo de certas operações geográficas. Como suposições básicas, admite-se que:

- os mapas estão representados vetorialmente;
- além do custo de acesso aos dados, o custo da execução de operadores espaciais complexos constitui um gargalo no desempenho global das operações geográficas;
- o custo de comunicação entre os processadores é negligível, quando comparado com o custo de acesso a memória secundária e o custo das operações locais (o que é o caso por exemplo de um *cluster* de estações de trabalho).

O enfoque adotado é uma variante de uma estratégia genérica que consiste em distribuir, de forma balanceada, o armazenamento entre processadores, permitindo a paralelização das operações. As peculiaridades pertinentes a mapas e operações geográficas vêm à tona em dois aspectos. Primeiro, o critério de distribuição entre processadores adotado obriga a divisão de mapas em fragmentos e leva em conta apenas a posição geográfica dos objetos. Isto significa que feições da superfície terrestre que estão geograficamente próximas tenderão a ter suas representações armazenadas no mesmo processador. Ou seja, proximidade geográfica induz proximidade física de armazenamento, o que representa uma heurística adequada para minimizar o custo das operações geográficas. Segundo, a compilação de operações geográficas sobre mapas em operações locais sobre os dados armazenados em cada processador exige uma análise cuidadosa da semântica das operações de forma a evitar que a distribuição dos objetos induza erros semânticos.

1.1 Objetivo do Trabalho

O objetivo da dissertação é propor e analisar *qualitativamente* um método de armazenamento distribuído que introduza o uso de paralelismo na resolução de consultas espaciais. O foco do trabalho é o tratamento das representações de mapas temáticos e cadastrais sob a forma vetorial.

A estrutura proposta atende aos seguintes requisitos:

- eficiência no acesso às informações (consulta por varredura);
- eficiência na resolução de operações entre conjuntos de dados (consultas por junção espacial);
- uso de métodos de acesso e algoritmos já existentes.

Além de propor e avaliar um método de armazenamento distribuído (capítulo 4), a dissertação ainda deve discutir formas de uso da estrutura no contexto da resolução de operações geográficas. Não se pretende, nesse caso, cobrir por completo o rol de operadores possíveis, nem construir um interpretador ou otimizador de consultas, mas propor um método para a síntese de algoritmos que implementam essas operações.

1.2 Caracterização do Problema - Sistemas de Informação Geográfica

Um Sistema de Informação Geográfica (SIG) é um conjunto de ferramentas integradas usadas para abstrair e modelar dados geográficos [Mor92]. São usados na captura, armazenamento, conferência, manipulação, análise e apresentação de dados espacialmente referenciados [RM92] em conjunto com dados convencionais a eles relacionados. Ou seja, um SIG usa dados georeferenciados para descrever elementos e fenômenos do mundo real associados a sua localização na superfície terrestre. Um SIG é um sistema de informação aplicado a dados geográficos; um conjunto de processos que, a partir do dado cru, original, armazena e produz informações para gerência, análise e tomada de decisões no universo espacial.

A importância dos Sistemas de Geoprocessamento pode ser avaliada a partir de suas aplicações típicas para análises sócio-economicas, ambientais e de gerenciamento cadastral. A unidade básica de interpretação espacial no modelo e análise do mundo é o mapa. Por sua vez, um mapa deve ser entendido como um dispositivo para o armazenamento de informações espaciais que representa, em uma superfície plana e em escala menor, dados de terrenos, estados, regiões, países, etc... [Cif95]. Um mapa é constituído de objetos geográficos, entidades estáticas com localização fixa em relação à superficie terrestre e descritos por sua geometria.

A dificuldade na manipulação manual desses dados motivou inicialmente o desenvolvimento de sistemas de geoprocessamento. O desenvolvimento e integração de diversas tecnologias permite, hoje, que sejam diponibilizados uma boa quantidade de informação em excelentes níveis de precisão para análise do meio físico em que vivemos e na forma em que o homem e as diversas entidades do ecossistema interagem entre si.

A complexidade de um SIG encontra-se no fato dele integrar informação espacial e não-espacial num só ambiente, oferecendo um ambiente consistente para a análise de dados geográficos. A integração de mapas e outras formas de informação em meio digital e acessíveis através de ferramentas eficientes de manipulação e análise permite um novo enfoque no desenvolvimento de aplicações em diversas áreas, baseadas em cartografia, mapeamento, gerência de recursos naturais, agricultura, planejamento urbano e regional, etc...

Os SIGs hoje existentes podem ser definidos como bancos de dados específicos para informações espaciais (SGBD não convencional) associados a conjuntos de operações que manipulam esses dados (funções analíticas) e a ferramentas que manipulam dados convencionais. Em resumo, caracterizam-se por integrar em uma única base de dados, informações cartográficas, dados de censo e cadastro urbano e rural, imagens de satélite, redes e modelos numéricos de terreno. Combinam os mais diversos tipos de informação através de algoritmos de manipulação para gerar mapeamentos derivados e, também, consultam, recuperam e visualizam o conteúdo da base de dados [INP94].

Dentro deste universo, diversas áreas de pesquisa têm contribuído para o desenvolvimento e aperfeiçoamento de tais ferramentas com o intuito de tornar o uso potencial de SIGs uma realidade. Uma das formas de decompor a funcionalidade de tais sistemas é pelo tipo de dado ao qual cada conjunto de operações/algoritmos é aplicado.

Numa visão orientada a banco de dados, pode-se definir um SIG como um SGBD não convencional que disponibiliza um conjunto de processos especializados no tratamento de dados espaciais. Um banco de dados geográfico manipula uma grande quantidade de dados dos mais diversos formatos, entre eles os formatos vetorial (por exemplo, representando mapas temáticos e cadastrais), varredura (imagens e matrizes de valores), amostras 3D (altimetria e modelos numéricos de terreno), grades regulares (topografia e modelos numéricos de terreno) e triângulares (relevo). Um SIG, então, deve oferecer funcionalidade para armazenamento e manipulação eficiente de todos esses tipos de dados.

Em termos de funcionalidade, um SIG deve possuir capacidade cartográfica para coleta de dados, mapeamento e geração de cartas; capacidade de gerenciamento de dados, constituída por edição, processamento de consultas, otimização, gerência de transações, controle de acesso e segurança, indexação e modelagem de dados; e capacidade analítica, oferecendo poderosas ferramentas através de operações escalares (distância, área...), topológicas (que checam relacionamentos entre objetos) e espaciais (que retornam novos objetos geográficos), tais como superposição de polígonos, proximidade e busca espacial.

Por exemplo, os métodos de acesso devem fornecer caminhos otimizados para recuperação de dados a partir de seus atributos espaciais ou convencionais. Um otimizador de consultas avalia a melhor estratégia para satisfazê-las. Um modelo de dados deve ser expressivo, permitindo a representação das várias formas de informações através de dados espaciais e convencionais.

Cada um desses assuntos constitui um tema para investigação científica para incrementar e oferecer modelos de dados expressivos, linguagens de consultas apropriadas e mecanismos **eficientes** para execução de operações sobre dados espaciais. O foco deste trabalho é contribuir para a eficiência da manipulação de dados vetoriais, normalmente caracterizado por uma geometria complexa, já que a representação vetorial se propõe a representar objetos de uma forma precisa em termos de posição, dimensão e tamanho.

1.3 Estrutura do Trabalho

Além desse primeiro capítulo introdutório que descreveu o objetivo da dissertação, caracterizando os problemas relacionados a área de Sistemas de Informações Geográficas, o texto desenvolve-se segundo a estrutura a seguir.

O capítulo 2 descreve a motivação da dissertação nas áreas de Paralelismo e Banco de Dados. Aborda, ainda, problemas e possíveis soluções nas áreas de particionamento de dados, balanceamento de carga e concentração dos dados/consultas em parte do sistema paralelo (*data skew*). E finalmente, discute alguns trabalhos correlatos.

O capítulo 3 discorre sobre o modelo para dados geográficos usado e define os conceitos de mapa, seus elementos e sua fragmentação. Discute, em seguida, uma arquitetura para um gerenciador de objetos geográficos e suas possíveis extensões num ambiente paralelo. Enfoca, com mais detalhes, o módulo vetorial desse ambiente descrevendo possíveis organizações do banco de dados.

Já o capítulo 4 descreve a principal contribuição deste trabalho, composta da definição de estruturas de dados espaciais sobre a arquitetura paralela definida anteriormente. Introduz métodos de armazenamento distribuídos, incluindo a definição das estruturas de dados, rotinas de edição e considerações sobre o catálogo de dados e o balanceamento de carga.

A última parte da dissertação trata do uso das estruturas propostas, ou seja, da definição de algoritmos para a resolução de um conjunto básico de operações sobre a estrutura. O capítulo 5 descreve os operadores estudados, apresentando sua definição e a descrição de procedimentos para resolvê-los a partir das estruturas de dados propostas. Os operadores podem, a princípio, ser classificados em dois sub-grupos: operadores que têm como parâmetro apenas um mapa e os que envolvem mais de um mapa. Ambos são analisados em função de suas características comuns e específicas.

Finalmente, o capítulo 6 apresenta as considerações finais, ressaltando as contribuições da dissertação e propondo direções para trabalhos futuros.

Capítulo 2

Preliminares

Como foi descrito no Capítulo 1, diversas áreas de pesquisa motivaram esta dissertação. Este capítulo descreve de forma geral cada uma das áreas de interesse, incluíndo referências a trabalhos publicados.

A seção 2.1 discorre sobre a área de computação paralela, discutindo o desenvolvimento das arquiteturas convencionais em direção às arquiteturas paralelas, dando ênfase ao modelo de memória distribuída, ambiente usado na arquitetura proposta pela dissertação. Esta seção apresentará, ainda, os modelos de programação paralela usuais.

A seção 2.2 introduz a área de Bancos de Dados Paralelos, especificamente os que fazem uso de arquiteturas do tipo *shared-nothing*. Discorre, também, sobre as diferenças entre bancos de dados distribuídos e paralelos.

A seção 2.3 faz algumas considerações sobre aspectos típicos da distribuição de dados num sistema de banco de dados paralelo, especificamente sobre o particionamento de dados, o balanceamento da carga pelo sistema e a concentração de dados ou consultas em determinadas regiões do banco de dados.

Finalmente, a seção 2.4 discute uma série de trabalhos correlatos.

2.1 Paralelismo

Ao longo da história da computação, o desenvolvimento da tecnologia disponível foi imenso. Os equipamentos estão cada vez mais rápidos, com maior capacidade de armazenamento e mais baratos. Esse rápido desenvolvimento, porém, não tem atendido a demanda por desempenho que as aplicações vêm apresentando.

Esta seção apresenta a motivação no estudo de arquiteturas paralelas como uma alternativa para os sistemas de computação convencionais. Na sub-seção 2.1.1, as arquiteturas paralelas são apresentadas a partir da descrição das limitações de uma máquina convencional. A seção 2.1.2 define algumas métricas para a avaliação de arquiteturas e sistemas paralelos. Na seção 2.1.3, são descritos os modelos de arquitetura paralela mais comuns, o de memória distribuída (ambientes *share-nothing*), escolhido como arquitetura básica usada no trabalho aqui apresentado e o de memória compartilhada (*shared-memory*). Finalmente a última seção (2.1.4) faz um breve resumo sobre os modelos de programação mais comuns num ambiente paralelo.

2.1.1 Arquiteturas Convencionais x Arquiteturas Paralelas

A arquitetura, forma em que os componentes de uma máquina são organizados, mais encontrada nas últimas décadas é baseada no modelo de *Von Neumann* [ITS94]. Este propõe um sistema que integra: memória, onde são armazenados instruções e dados; uma unidade de controle e processamento, que executa um determinado conjunto de instruções; e um barramento que faz a comunicação entre os dois. Um relógio é usado para sincronizar o funcionamento dos diversos elementos.

Neste ambiente, um programa (presente na memória) é executado de forma seqüencial, instrução a instrução. Cada uma dessas é mapeada numa função a qual o processador é capaz de executar. Em arquiteturas do tipo CISC (*Complex Instruction Set Computers*), o conjunto de instruções disponível é próximo ao das linguagens de alto nível. A idéia é transferir o máximo possível das operações para o *hardware* que a princípio seria mais eficiente.

Por outro lado, as máquinas RISC (*Reduced Instruction Set Computers*) provêm um conjunto bem menor de instruções primitivas, mais simples, mas que são executadas sempre num número fixo de passos. Esta característica permite que a estrutura da unidade de processamento seja mais eficiente, permitindo ganhos consideráveis em relação ao outro tipo de máquina. Com o advento de tecnologias que permitiram a construção de máquinas RISC mais baratas como por exemplo, *caches* que aumentam a velocidade do *fetch* de instruções, diminuição dramática da relação tamanho/preço de memória, *pipelining* mais eficientes e de compiladores mais avançados [Dow93], este modelo de arquitetura vem ganhando cada vez mais espaço no mercado.

O aumento na velocidade dessas máquinas, porém, parecem estar chegando a um limite intrínseco ao modelo seqüencial imposto pela arquitetura de Von Neumann. Por outro lado, o desempenho de aplicações, em especial na área de banco de dados, vem se tornando um fator crítico por envolverem consultas cada vez mais complexas e bases com grandes volumes de dados [Mat93].

Surge, então, a idéia de computação paralela que usa múltiplos processadores para minimizar o tempo para a resolução de tarefas e processos. O requisito básico para permitir a computação paralela é a definição de um *hardware* que componha vários processadores e um meio de comunicação entre eles.

Uma arquitetura paralela é caracterizada por:

- seu modelo de memória. Os dois tipos de modelo mais comuns são os chamados memória compartilhada (*shared-memory*) e memória distribuída (*shared-nothing*). A descrição de cada um deles e uma pequena comparação é feita na seção 2.1.3.
- sua topologia, que define a forma em que os nós (ou processadores) do sistema são interligados, ou seja, o mecanismo de comunicação entre processadores. A mais simples estratégia de conexão é a chamada *lattice* [Sny95], onde cada um dos nós é ligado diretamente aos seus *vizinhos*. Um outro mecanismo muito encontrado chama-se *hypercube* [Sny95] definido por um número de nós potencia de dois, identificados por um código binário único. Cada nó tem ligação direta com os nós cujo identificador diferencia do dele em apenas um dígito na representação binária. Finalmente, existem os *switches* (por exemplo, SP2 [SSA+94]) que conectam pares de processadores quando necessário. A conexão, nete caso, é direta, mas "ligada" em tempo real.
- número de nós da conexão, que pode variar entre poucas unidades até milhares de processadores. Note que arquiteturas que compartilham recursos têm nessa

característica um ponto de estrangulamento e podem não ser capazes de crescer indefinidamente. A capacidade de crescimento de um sistema é chamada *escalabilidade*.

- tipo de seus processadores. Um sistema paralelo pode ser homogêneo ou heterogêneo.

- seu modelo computacional, que varia entre SIMD (*Simple Instruction, Multiple Data*) e MIMD (*Multiple Instructions, Multiple Data*). O primeiro define um único código executado sobre diferentes conjuntos de dados em paralelo, enquanto o segundo executa diferentes trechos de código em diferentes conjuntos de dados.

- seu modelo de programação paralela, que pode pressupor o compartilhamento de memória (*shared-memory*) ou a comunicação entre processos exclusivamente através de troca de mensagens (*message-passing*). Apesar de cada um desses modelos serem adequados mais especificamente a um dos modelos de memória, eles podem ser simulados em ambientes cujo *hardware* difere do modelo usado. Por exemplo, a troca de mensagens pode ser simulada em memória compartilhada. As características desses modelos estão descritos em 2.1.4.

2.1.2 Métricas

O processamento paralelo pode ter dois objetivos: (1) aumentar a disponibilidade; ou (2) reduzir o tempo de resposta da aplicação [LDSY91]. A partir desses objetivos, define-se duas métricas que avaliam um modelo paralelo. Um sistema paralelo ideal possui como propriedades o *speed-up* e o *scale-up* lineares [DG90].

O *speed-up* S_e é definido pelo fator:

$$S_e = \frac{T_p}{T_g}$$

onde T_p é a duração de um dada tarefa executada num sistema pequeno e T_g, a duração da mesma tarefa num sistema grande [DG92]. A taxa é linear, se um sistema N vezes maior tem uma taxa de *speed-up* N vezes maior. Ou seja, ao dobrar o número de processadores que executam uma tarefa, ela é executada na metade do tempo.

O *scale-up* S_c é definido pelo fator

$$S_c = \frac{T(S_p, P_p)}{T(S_g, P_g)}$$

onde $T(S_p, P_p)$ é o tempo que um sistema pequeno leva para executar um problema pequeno e $T(S_g, P_g)$ é o tempo que um sistema grande leva para executar uma tarefa grande. O *scale-up* é linear se a taxa acima for um, ou seja, se duas vezes mais processadores executa duas vezes mais tarefas no mesmo tempo.

Uma terceira medida é a *eficiência* do sistema definida pela percentagem dos processadores disponíveis para a resolução de um dado problema numa máquina dedicada [Sny95].

Os fatores básicos que levam a uma variação não linear dessas métricas são:

- *Inicialização* (*startup*), o tempo necessário para iniciar a execução em paralelo. Normalmente associada a distribuição de tarefas e/ou dados pelos nós do sistema.

- *Interferência*, a perda de eficiência do sistema pela interação entre nós. Quanto mais recursos compartilhados (inclusive dados), maior tende a ser a interferência.

- *Distribuição*(*skew*), se na média o tempo de execução das tarefas diminui, mas alguns passos aumentaram muito, o paralelismo pode não melhorar o desempenho da aplicação como um todo.

2.1.3 Modelos de Memória

Como foi dito em 2.1.1, uma arquitetura paralela pode implementar diferentes modelos de memória, definidos a partir do modo de compartilhamento de memória primária e secundária. Os mais encontrados são (1) o modelo de memória compartilhada (*shared-memory*) e o de memória distribuída (*shared-nothing*). A referência [Gra93] cita, ainda, uma arquitetura híbrida com grupos de nós com memória compartilhada interligados numa arquitetura de memória distribuída. Este modelo incorpora tanto as vantagens como as desvantagens dos dois outros.

Um sistema de memória compartilhada é constituído por uma série de processadores que compartilham uma única memória global e um único disco comum. Cada um dos nós pode ou não ter áreas próprias de armazenamento ou *cache*. Já um sistema de memória distribuída, também chamado de sistema paralelo fracamente acoplado, é composto por uma série de nós independentes que possuem sua própria memória e unidade de armazenamento locais e são interligadas por um meio de comunicação pelo qual trocam mensagens.

As arquiteturas de memória compartilhada apresentam alto grau de paralelismo, pois os dados estão disponíveis a todos os processadores, porém, a taxa de interferência é alta, pois os recursos compartilhados representam um gargalo para o desempenho do sistema.

Sistemas de memória distribuída são encontrados tanto em máquinas paralelas (SP2) como em configurações de *clusters* de estações de trabalho ligadas por redes de alto desempenho. Esta última vem sendo usada como alternativa para os supercomputadores ou máquinas de alto desempenho.

A expansibilidade dessa alternativa é bem maior, pois evita compartilhar recursos que se tornam gargalos de desempenho com o crescer do sistema (adicionando mais processadores). Ela também aumenta a disponibilidade do sistema, pois podem ser implementados mecanismos de isolamento de falhas. A programação, porém, neste ambiente é mais complexa, pois exige o controle dos dados compartilhados e replicados e o controle de concorrência.

2.1.4 Modelos de Programação Paralela

A síntese de algoritmos num ambiente paralelo é normalmente orientada ao modelo de compartilhamento de memória usado. Apesar disso, algoritmos desenvolvidos para uma arquitetura são facilmente adaptados para a outra a partir de um ambiente que simula memória compartilhada ou troca de mensagens em arquiteturas que não implementam esses recursos diretamente.

O modelo de memória compartilhada é naturalmente desenvolvido a partir do paradigma de que os diversos processadores têm acesso aos mesmos dados e é oferecida, por exemplo, pelo ambiente Linda [CGMS94]. Neste modelo, as aplicações podem manter uma única estrutura de dados na memória compartilhada, que é acessada e atualizada por processos em qualquer um dos nós do sistema. Um ambiente desse tipo deve definir a unidade e forma de bloqueio da estrutura compartilhada. O compartilhamento de recursos pode, porém, representar um gargalo sério no desempenho do sistema, se a estrutura de dados for continuamente bloqueada por processos concorrentes.

Por outro lado, modelos definidos por troca de mensagens fazem uso de recursos locais independentes e se comunicam com os demais processos quando necessário. Um exemplo de ambiente que implementa este modelo é o PVM [SGRM94]. Os diferentes nós do sistema cooperam entre si através de mensagens que podem ser enviadas de qualquer nó para qualquer outro nó do sistema a partir da definição de um endereçamento único.

Podem ser definidas, ainda, duas estratégias gerais para a programação paralela [Sny95]. A primeira delas paraleliza as instruções (*control parallel*) a serem executadas, enquanto a segunda executa uma mesma instrução sobre diversos conjuntos de dados distribuídos. A forma em que os dados são particionados e alocados aos diferentes nós do sistema se torna condição essencial para a eficiência do sistema.

Neste ambiente pode-se imaginar uma arquitetura mestre-escravo, onde um processador funciona como o mestre que distribui tarefas e/ou dados pelos diversos processadores escravos. O nó mestre é visto como uma interface centralizada da aplicação com os recursos distribuídos do sistema.

Outro paradigma comum é o cliente-servidor que apresenta uma configuração onde um ou mais nós do sistema funcionam como provedores de serviços tanto de armazenamento como de processamento. Os nós clientes requisitam um serviço do servidor quando necessário, este executa o serviço e devolve uma resposta ao cliente.

O modelo de programação paralela utilizado no texto implementa o conceito de troca de mensagens entre os nós do sistema como forma de compartilhamento de recursos do sistema.

A figura 2.1 resume o uso das primitivas e comandos usados no texto.

A primitiva *send* envia ao processador identificado por proc, um comando definido como segundo parâmetro mais uma lista de dados que contém os argumentos do comando. Essa primitiva é implementada sobre serviços de rede confiáveis, onde há garantia do recebimento da mensagem pelo destino de forma transparente a aplicação.

Primitivas:

send(proc, comando, lista_dados)
receive(proc, comando, lista_dados)
wait until (condição)

Comandos:

store (dados)
move (dados)
get (M, p_A, p_B)
read
transfer (c, destino)

Figura 2.1: Definição das primitivas e comandos utilizados nos algoritmos em pseudo-código

A primitiva *receive* lê do *buffer* de origem proc o comando a ser executado e seus respectivos parâmetros (lista_dados). Essa primitiva não bloqueia a execução do processo, retornando valores nulos se não houver informação no *buffer* a ser lida.

A primitiva *wait until* bloqueia o processo a espera de uma condição.

O comando store sinaliza a execução da rotina de armazenamento do método de acesso local.

O comando move armazena dados no processador destino e em seguida remove-os do nó origem.

O comando get acessa os dados do fragmento do mapa M definido pelo retângulo (p_A, p_B).

O comando read lê o conteúdo do *buffer* local.

O comando transfer sinaliza a execução de uma transferência da célula de fragmentação c para o nó destino.

2.2 Banco de Dados

Nos primórdios do processamento de dados, cada aplicação era responsável pela aquisição e gerência das informações com as quais lidava. O conceito de Banco de Dados surgiu com o desenvolvimento da idéia de independência dos dados, os quais podem ser acessados por diversas aplicações e serem gerenciados por um sistema próprio, o Sistema de Gerência de Banco de Dados (SGDB). O objetivo de um banco de dados é, então, a integração dos dados.

No início dos anos 80, surgiram esforços de desenvolvimento de máquinas paralelas especializadas para banco de dados, as Máquinas de Banco de Dados (*database machines*). Esses esforços foram duramente criticados, já que a maior parte das pesquisas realizadas na época visavam o desenvolvimento de *hardware* especializado baseado em tecnologias (*bubble or CCD memory* , discos óticos, discos *head-per-track*) que não alcançaram os níveis de desempenho prometidos [DG92]. Com o decorrer do tempo, viu-se que as máquinas de banco de dados, com arquiteturas de *hardware* exóticas, realmente não progrediram, mas o uso de sistemas de bancos de dados paralelos vem se proliferando. Isso ocorreu em parte devido à primazia que os bancos de dados relacionais representam hoje no mercado (o modelo de consulta relacional se adequa perfeitamente a ambientes paralelos) e pelo desenvolvimento das arquiteturas paralelas em geral.

A utilização de arquiteturas paralelas na hospedagem de banco de dados visa o aumento do desempenho nas aplicações típicas da área quando envolvem grandes massas de dados e operações complexas. O processamento de operações oferece várias oportunidades para o uso de paralelismo, mas é estrangulado pelo acesso aos dados. Diante disso, a experiência adquirida na gerência de banco de dados distribuídos têm sido largamente aproveitada no que se refere principalmente ao armazenamento e acesso paralelo e ao processamento paralelo desses dados [Mat93].

A tecnologia de Sistemas de Banco de Dados Distribuídos (SBDD) se originou da união de esforços nas áreas de Banco de Dados e Redes de Computadores. Usualmente, em ambientes tradicionais, a integração dos dados é feita de forma centralizada, o que facilita o controle e a gerência dos mesmos. Esse fato era verdade no passado, quando não estavam disponíveis estações de trabalho poderosas ou tecnologia de comunicação de dados eficientes. Devido à importância da tecnologia de banco de dados distribuídos na construção de tecnologias de bancos de dados paralelos, faz-se a seguir uma breve descrição sobre ela.

Os SBDDs são sistemas que propõem solucionar o limite de desempenho imposto por gargalos nas operações de E/S em uma máquina convencional. Eles visam explorar o paralelismo inerente a muitas aplicações e também refletir a descentralização do gerenciamento dos sistemas. Desta forma, é possível listar as vantagens no uso de SBDD, o que será feito a seguir tomando como base a referência [OV91].

Um SBDD possui *autonomia local*. O processamento local é executado independentemente das condições do sistema geral e a responsabilidade de gerência dos dados é distribuída com eles.

Têm-se uma *melhoria no desempenho*. Isso ocorre por parte do processamento ser local, dos dados de maior interesse estarem armazenados localmente ou mesmo, pelo fato de processamento e dados estarem distribuídos de modo mais uniforme no sistema, permitindo que recursos ociosos no ambiente sejam melhor aproveitados. É possível, também fazer uso do paralelismo inerente a algumas aplicações.

Além disso, existe a possibilidade de dados essenciais serem replicados em mais de um nó do sistema, permitindo que sejam acessados mesmo com a falha de um ou mais nós, discos ou parte do próprio meio de comunicação, aumentando a *disponibilidade do sistema*.

Com o desenvolvimento de estações de trabalho e redes cada vez mais baratas, a manutenção de um sistema formado pela composição de diversos componentes menores tornou arquiteturas paralelas bem mais interessantes financeiramente do que o uso de máquinas de grande porte convencionais. Outro aspecto é dado pelo custo de comunicação no caso de bancos de dados geográficamente dispersos. Se as informações gerenciadas por um nó do sistema estão armazenadas localmente, a tendência é que o tráfego na transferência das mesmas seja bem menor.

A arquitetura desses sistemas costumam permitir fácil crescimento de acordo com a demanda de desempenho e a capacidade de armazenamento (*expansibilidade*) durante a história do desenvolvimento do sistema.

No caso de sistemas em que os dados já são necessariamente distribuídos (por motivo geográfico ou organizacional), mas um SBDD não está disponível, esse passa a disponibilizar acesso antes inexistente a dados federativos.

Com todas essas vantagens porque, então, a tecnologia de banco de dados distribuídos não domina o mercado? Existe, é claro uma série de desvantagens em seu uso. Em primeiro lugar, a *tecnologia* usada é *recente*, fazendo com que a falta de experiência limite a disponibilidade de ferramentas e aplicações que permitam o bom aproveitamento da potencialidade desse tipo de ambiente.

Além disso, o *grau de complexidade* inerente a um sistema distribuído, seja ele homogêneo ou heterogêneo é muito maior que o de um ambiente convencional. A complexidade se dá tanto a nível de projeto, como de gerência e controle do sistema. Há, inclusive, uma série de problemas a serem solucionados que não tem nem paralelo num ambiente convencional equivalente, como por exemplo a distribuiç ao de carga pelo sistema e o controle de concorrência a recursos compartilhados.

O *custo*, que foi descrito como ponto positivo, pode funcionar também como aspecto desvantajoso. Em alguns ambientes, onde não havia necessidade de redes de comunicação, o custo do *hardware* e *software* complementar pode aumentar bastante. Da mesma forma, o desenvolvimento de *software* paralelo, por ser tecnologia recente, não totalmente dominada, tende a ser mais caro. O maior incremento de custo se dá (no caso de redes geográficamente distribuídas) pela necessidade de manter mais de uma equipe de suporte e manutenção, dado que a administração do sistema já não é mais centralizada. Isso pode acarretar em replicação de custos, inerente a gerências descentralizadas.

A *distribuição do controle* também pode se tornar uma desvantagem tornando obrigatória a definição de protocolos de gerência de responsabilidade. A distribuição do controle, muitas vezes, cria problemas de coordenação e sincronização no funcionamento de aplicações globais.

Outro aspecto a ser considerado é o da *segurança*. Os níveis de segurança obtidos em máquinas de grande porte estão muito acima dos conseguidos em ambiente de mini e micro computadores e nos meios de comunicação de dados.

Finalmente, um fator a ser considerado ainda é a *dificuldade de mudança* . Em geral, já houve pesado investimento na organização de bancos de dados centralizados e o custo da migração das aplicações e o re-treinamento dos usuários pode ser alto.

Entre outros problemas em aberto, o projeto de um SBDD analisa formas de fragmentação e distribuição dos dados, além da necessidade da replicação ou não de parte do banco. As mesmas soluções propostas neste caso devem ser aproveitadas nos bancos de dados paralelos.

A fragmentação dos dados é feita a partir do conhecimento prévio dos dados ou da aplicação que os processa. Esta fragmentação, num banco de dados relacional, pode ser feita em cortes horizontais, verticais ou híbridos nas tabelas existentes. A alocação pode permitir ou não a existência de replicação de dados em diferentes nós do sistema. Quando não há nenhuma replicação, diz-se que o banco de dados é totalmente particionado. Para uma melhor alocação é necessário novamente o conhecimento dos dados, das aplicações, de informações sobre o nó e das condições da rede.

O processamento de consultas distribuídas ou paralelas tem particularidades sob o aspecto da busca e acesso aos dados (que estão distribuídos) e nas formas de execução da consulta que podem ser *interquery*, *intraquery* (ou *interoperator*) ou *intraoperation*. Como a própria nomenclatura indica, a paralelização da execução de consultas pode ser efetivada distribuindo, respectivamente, as diversas consultas, os diferentes predicados de uma consulta ou a execução propriamente dita de cada operação.

2.2.1 BD distribuídos X BD paralelos

Na seção anterior, foi apresentado um panorama geral dos tópicos estudados na área de banco de dados distribuídos. Tal apresentação visou ser, na medida do possível, generalista. Pode-se fazer, porém, distinção entre banco de dados distribuído e paralelo. Essa seção definirá cada um dos tipos de ambiente, destacando as características típicas de um e de outro que podem alterar decisões de projeto do banco de dados.

Uma das mais evidentes distinções a ser considerada no desenvolvimento de ferramentas e aplicações nesses sistemas se refere ao desempenho no acesso a informações locais e remotas. É claro que o mesmo é diretamente dependente da arquitetura sobre a qual está implementado o sistema, mas é possível fazer uma avaliação de forma geral. A príncipio, pode-se considerar que um banco de dados distribuído, muitas vezes geograficamente disperso, possui um custo de comunicação de dados maior que o de um banco de dados paralelo. Este último, devido ao seu alto desempenho na execução de consultas, pode ser usado como servidor de dados em um banco de dados distribuído.

Em outras palavras, a excessiva troca de dados entre nós num banco de dados distribuído, em geral, deve ser evitada, privilegiando o processamento local quando possível. Por outro lado, esse mesmo custo pode ser desconsiderado no caso de ambientes paralelos. Com o uso de arquiteturas de comunicação extremamente rápidas, torna-se praticamente indiferente o acesso a informações alocadas localmente ou em outros nós do sistema.

Um banco de dados distribuído, normalmente, implica em autonomia local [Gra93]. Cada nó do sistema funciona de forma independente, sendo responsável pelo controle do acesso aos seus próprios dados, gerenciamento do processamento de consultas local, etc... Ou seja, num banco de dados distribuído, o funcionamento de cada nó independe das condições dos demais. Esses sistemas são os que apresentam vantagens vísiveis na autonomia local, disponibilidade e compartilhamento de dados federativos (ver seção 2.2).

Um banco de dados paralelo ou alguns SGBD distribuídos homogêneos só possuem um sistema gerenciador de banco de dados, responsável pela divisão das consultas em fragmentos e a execução dos mesmos em paralelo. O acesso aos dados, nesse caso, ainda pode ser independente, feito pelo nó que efetivamente aloca os mesmos. O controle do acesso, porém, tende a ser independente de onde realmente o dado é armazenado. O processamento de consultas, nesses sistemas, normalmente considera todos os nós disponíveis e o seu uso é controlado por um gerenciador global.

Ambos os tipos de sistema podem ser construídos sobre diversos tipos de arquitetura. Um banco de dados distribuído pode ser homogêneo (todos os nós possuem *hardware* e *software* equivalentes) ou heterogêneo (ambiente operacional - *hardware* e *software* - em cada nó pode variar, se comunicando por interfaces comuns) e pode também empregar algoritmos paralelos na execução de consultas, aproveitando a disponibilidade de CPU no sistema.

Os ambientes de banco de dados paralelos podem ser baseados em arquiteturas de memória compartilhada, discos compartilhados, memória distribuída ou arquiteturas hierárquicas, constituídas de múltiplos *clusters* independentes, cada um dos quais possuindo grande memória compartilhada.

Essa última é considerada em [Gra93] ideal para o desenvolvimento de novos sistemas, pois é considerada mais genérica. Generalidade, porém, nem sempre é um bom critério de escolha, já que tende a apresentar, até mesmo de forma acentuada, os problemas típicos, nesse caso, tanto de ambientes de memória compartilhada, quanto de memória distribuída. Cada um desses ambientes apresentam vantagens e desvantagens. Em [Sto86], apresenta-se o caso de memória distribuída como o mais promissor como plataforma para bancos de dados distribuídos e paralelos.

A próxima seção discorre um pouco mais sobre as características dessa arquitetura (*shared-nothing*), ambiente a ser considerado nos demais capítulos da dissertação.

2.2.2 Arquitetura *Shared-Nothing*

Sistemas de banco de dados em arquiteturas de memória distribuída (*shared-nothing*) distribuem a base de dados pelos diferentes processadores, cada um possuindo memória e discos locais próprios. Essa arquitetura minimiza a interferência dos nós entre si, minimizando o compartilhamento de recursos [DG92].

Com a distribuição dos dados é possível executar localmente filtros prévios e apenas um sub-conjunto limitado aos dados de possível interesse trafegam no meio de conexão permitindo um projeto escalar.

Exemplos de sistemas de bancos de dados que usam essa arquitetura são o Teradata, Gamma, Tandem, Bubba, Arbre e nCUBE, apesar de cada um deles ser construído sobre uma topologia de comunicação diferente.

A principal vantagem da arquitetura é permitir seu crescimento até centenas ou milhares de nós.

Alguns problemas, porém, devem ser cuidadosamente estudados para permitir um bom desempenho na prática desses sistemas de banco de dados. Deve ser definido um bom critério para o particionamento dos dados que permita um balanceamento de carga uniforme mesmo em ambientes em que se apresenta uma concentração de dados em uma determinada região do domínio.

2.3 Aspectos da Distribuição de Dados

Essa seção aborda detalhadamente os aspectos estudados pela dissertação nas áreas de particionamento dos dados, balanceamento de carga e concentração parcial dos dados.

Na seção 2.3.1, além de apresentado o problema do particionamento de dados, são estudadas as políticas usuais utilizadas em bancos de dados convencionais. É feita, ainda, uma extensão do problema para o caso de dados espaciais e a análise da adequação das soluções existentes para os mesmos.

Mesmo considerando uma partição razoável dos dados em que o domínio seja dividido em conjuntos de tamanho e complexidade equivalentes, a alocação de cada uma das partes pelos diversos nós do sistema não é trivial. Para um melhor aproveitamento do ambiente, o balanceamento de carga (considera-se tanto a quantidade de dados como o custo de processamento desses dados e heurísticas para a obtenção do mesmo são assuntos discutidos na seção 2.3.2. Nela, são apresentadas as políticas normalmente usadas, o suporte oferecido pela estrutura de dados e o relacionamento das possíveis soluções para esse problema com as políticas analisadas para o particionamento dos dados.

Algumas das estruturas analisadas nas primeiras seções deste capítulo podem ocasionar uma maior concentração de dados em determinados nós do sistema, o chamado problema de *data skew*. A seção 2.3.2 mostra, ainda, que, mesmo que seja obtido um balanceamento de carga perfeito, dependendo da organização proposta para o banco de dados, é possível que a maior parte das consultas efetivamente feitas privilegiem uma dada região, comprometendo o grau de paralelismo a ser alcançado (*load x workload*.

2.3.1 Particionamento dos Dados

Uma das formas de obter paralelismo, o chamado paralelismo particionado [DG92], é conseguida a partir do particionamento dos dados e sua distribuição pelos nós do sistema. Diversos estudos vêm mostrando que o desempenho e escalabilidade de um banco de dados paralelo é bastante dependente da forma com que os dados estão fisicamente armazenados nos diversos nós do sistema. Caso essa divisão e distribuição não seja feita de forma adequada, a execução de operadores sobre a massa de dados deixará recursos do sistema ociosos.

Com isso, o banco de dados deve ser particionado a partir de uma política adequada e cada uma dessas partes deve ser alocada a nós do sistema paralelo (a política de alocação não necessariamente é um mapeamento um para um do número de partições para o número de processadores - veja seção 2.3.2). Cada processador é, então, responsável pelo processamento local de seus dados. Em uma etapa posterior, os vários resultados parciais são compostos numa única saída. A figura 2.3.1 exemplifica o uso desse esquema num sistema de três processadores numa arquitetura de memória distribuída.

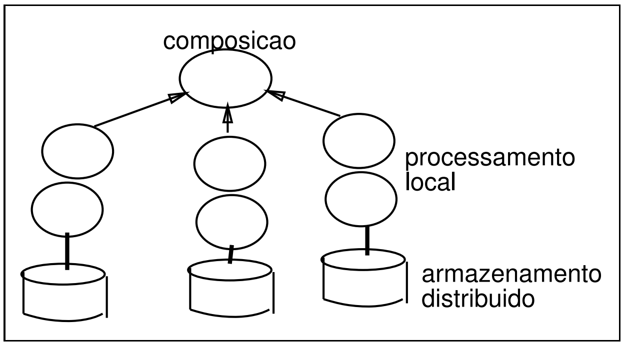

Figura 2.2: Processamento paralelo através de uma arquitetura de dados particionados

O particionamento dos dados permite que o SGBD faça uso da existência de mais de um sistema de entrada e saída presentes num ambiente paralelo. Num banco de dados relacional, pode-se encontrar diversas políticas de particionamento [DG92]. A seguir, descreve-se as formas básicas que podem ser usadas de modo independente ou integradas, gerando critérios mais complexos para a divisão dos dados.

A mais simples delas distribui as tuplas usando um critério de *round-robin*, ou seja, cada registro da tabela é colocado num disco/nó do sistema seqüencialmente. Essa política é perfeitamente adequada a buscas seqüenciais na tabela, pois o acesso é feito em todos os nós alternadamente. A maioria das consultas feitas, porém, usam alguma restrição para selecionar as tuplas de interesse. Essa primeira forma de particionamento não faz uso dessa característica, não podendo, portanto, ser aproveitada para tornar mais eficiente o acesso a resultados de consultas genéricas.

A partição dos dados pode ser feita, também, a partir da aplicação de uma função de *hash* a um dos atributos da relação. Dessa forma, os registros com esse atributo em comum se concentram num mesmo nó do sistema, minimizando o acesso a diferentes discos numa consulta restrita. Por outro lado, o uso de funções *hash* costuma distribuir registros de uma forma aleatória, diminuindo a proximidade de dados relacionados entre si. O reverso dessa característica, o agrupamento de um banco de dados, pode, em muitos casos, ser importante na otimização de acesso e operações sobre os dados.

Uma terceira forma de divisão do banco de dados leva em conta o agrupamento de tuplas similares numa mesma partição, é chamada *range partitioning*. Diversos algoritmos de agrupamento dos dados podem ser usados, permitindo um maior controle sobre a divisão dos dados. Uma das desvantagens no uso desse tipo de critério é a possibilidade de concentração de boa parte dos dados numa mesma partição ou a concentração da execução de consultas em um só nó do sistema. Esses problemas e possíveis soluções são analisadas na seção 2.3.2.

As políticas acima descritas consideram um particionamento horizontal das relações. Sua limitação está no fato de se basearem em apenas um atributo da relação. Consultas baseadas em outros atributos não são de nenhuma forma beneficiados da distribuição dos dados. Diversos estudos propõem estratégias a partir do uso de multiplos atributos [GDQ92, NHS84]. Esses critérios, em geral, se baseiam na combinação das políticas básicas aplicadas sobre diferentes atributos, segundo determinada prioridade avaliada a partir da semântica das relações.

Já no caso de banco de dados espaciais, pode-se considerar que o modo de indexação presente e também o atributo usado na formulação das consultas mais usuais é espacial. Logo, é razoável considerar que critérios espaciais serão usados também no particionamento dos dados. Esses critérios já são usados na definição das estruturas de dados usadas pelos métodos de acesso propostos na literatura nas áreas de bancos de dados espaciais e SIG [BH$^+$93, BKSS90, MCD94, Rez92, Sam90, NHS84].

Algumas dessas estruturas baseiam-se numa organização hierárquica dos dados, em forma de árvore, construída a partir da localização espacial dos objetos armazenados. Pode-se considerar que essas estruturas oferecem um particionamento natural dos dados, segundo um critério espacial bem definido. A figura 2.3 mostra um possível particionamento com o corte horizontal em um dos níveis de uma árvore genérica. Outras estruturas baseiam-se na definição de grades que fragmentam o espaço em regiões disjuntas. Cada uma desas regiões, também chamadas células da grade, definem um agrupamento de dados a ser alocado a um mesmo nó do sistema.

Um banco de dados espacial deve, então, ser particionado a partir de técnicas de indexação espacial e suas partes distribuídas pelos nós do sistema.

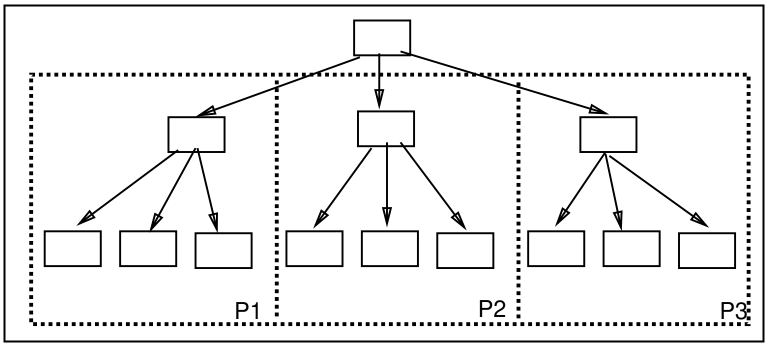

Figura 2.3: Partição dos dados a partir de uma árvore

2.3.2 Balanceamento e Concentração de Carga

As técnicas descritas na seção anterior descrevem a forma de organizar e distribuir o banco de dados. Elas, porém, não garantem uma carga de trabalho uniforme por todos os nós do sistema. Em outras palavras, a distribuição dos dados distribui a carga de processamento pelo sistema, mas por si só não oferece o balanceamento uniforme do mesmo.

Para obter bons índices de *speed-up* e *scale-up*, a carga de processamento deve ser distribuída cuidadosamente pelos processadores para garantir que o tempo de processamento em cada um deles seja equivalente [CK89].

Uma das dificuldades em obter um balanceamento de carga uniforme é o fato do conjunto de dados a ser armazenado não ser préviamente conhecido. A eficácia do balanceamento de carga depende também da arquitetura do sistema. Em arquiteturas de memória ou discos compartilhados, os dados não são necessariamente distribuídos e a carga de processadores é distribuída dinâmicamente. Já em uma arquitetura de memória distribuída, a distribuição da carga está intimamente ligada a distribuição de dados.

Ou seja, num ambiente em que o banco de dados é paralelizado a partir do particionamento dos dados, a função que os distribui é de extrema relevância. A função de particionamento dos dados deve, então, ter como principal preocupação a divisão da base de dados em partes equivalentes.

A característica de uma distribuição de dados que dificulta a obtenção do balanceamento requerido é a concentração dos dados numa dada região inerente a uma determinada amostra de dados. Tabelas de um banco de dados relacional podem apresentar em uma de suas colunas grupos de valores próximos ou idênticos que a partir de uma alocação baseada em taxas de valores (*range*) ou função *hash*, respectivamente são alocadas a um mesmo processador. No caso de banco de dados geográficos, também a distribuição efetiva de objetos no espaço pode configurar uma situação de maior concentração em algumas regiões.

A medida que o número de processos envolvidos no armazenamento dos dados cresce, maior é a influência deste problema, já que mesmo concentrações baixas podem influenciar o balanceamento de carga do sistema.

Duas políticas distintas guiam a solução da concentração de dados [Gra93]. A primeira delas, *skew avoidance*, visa evitar a concentração de dados na determinação prévia da função de distribuição da estrutura. Isso pode ser obtido, por exemplo, com o uso de estruturas de dados balanceadas cortadas e distribuídas num determinado ponto. O critério de alocação é dependente, neste caso, da distribuição efetiva dos dados garantindo a manutenção dinâmica do balanceamento.

A segunda política, *skew resolution*, reparticiona a base de dados toda a vez que percebe que o critério de distribuição usado não foi adequado. Sua implementação pode ser manual, deixando-a a cargo do administrador de banco de dados, ou automática, a partir da definição de uma série de heurísticas e do uso de estatística ou amostragem sobre o domínio de dados. Note que heurísticas implementadas dinâmicamente devem avaliar a opção entre manter a concentração de dados ou assumir o custo de sua redistribuição em tempo real.

Uma outra observação interessante se faz a partir do fato da a carga de processamento não estar balanceada mesmo em uma distribuição de dados ideal em que aparentemente não há nenhum tipo de concentração de dados, Essa situação ocorre quando, pela semântica dos dados armazenados, um determinado sub-conjunto do domínio é mais frequentemente solicitado.

Tomando novamente um banco de dados geográfico como exemplo, é possível que numa base de dados que cubra determinado tema em todo um continente, apenas os dados referentes a certos países ou regiões sejam continuamente consultados pelos usuários finais.

Esse fator é chamado concentração de consultas (*query skew*) e demonstra a grande importância da definição do modelo da base de dados. O interesse por diferentes conjuntos de dados pode variar, inclusive, no tempo de vida de uma aplicação, pois o usuário final pode aplicar uma abordagem em etapas na análise das informações armazenadas.

2.4 Trabalhos Relacionados

Esta seção descreve brevemente alguns trabalhos estudados que de alguma forma contribuíram na definição das estruturas propostas. Não se pretende analisar profundamente cada uma das estruturas (para tal, remete-se à bibliografia), mas ressaltar aspectos relevantes na caracterização dos conceitos utilizados na dissertação. Outros trabalhos estudados são referenciados ao longo do texto a medida em que são citados.

Sistemas de Geoprocessamento existentes empregam uma série de diferentes arquiteturas. Alguns deles armazenam todos os tipos de dados em um sistema de arquivo geral, outros empregam uma abordagem híbrida que manipula separadamente cada tipo de dado [Mor92] e finalmente, há uma abordagem integrada na qual todos os dados são armazenados num único SGBD [SR87].

O projeto PARADISE [DLPY93, DKL+94] propõe um sistema de informações geográficas num ambiente paralelo semelhante ao aqui considerado. Emprega, porém, uma arquitetura orientada a objetos que integra todos os tipos de dados num único sistema de banco de dados. Já a arquitetura considerada na dissertação emprega a abordagem híbrida, usando um sistema gerenciador de banco de dados convencional para o armazenamento de atributos não espaciais e métodos de acesso específicos para o armazenamento de representações espaciais.

Os demais trabalhos discutidos estão divididos em três assuntos. A seção 2.4.1 apresenta alguns métodos de acesso espaciais, em especial os usados como base na definição dos métodos de armazenamento propostos na dissertação. A seção 2.4.2 discute estruturas para armazenamento paralelo de dados espaciais. A seção 2.4.3 descreve alguns trabalhos que apresentam algoritmos diversos para a manipulação de estruturas vetoriais.

2.4.1 Estruturas de Acesso Espacial

A principal motivação para a definição de estruturas de acesso espaciais está em disponibilizar métodos para a recuperação de objetos espaciais de um conjunto grande desses objetos. Um método de acesso espacial organiza o espaço e os objetos nele inseridos de forma a facilitar sua recuperação. [LO93] faz um levantamento do desenvolvimento de métodos de indexação espacial, classificando-os a partir das técnicas de indexação tradicionais que os originaram. Também podemos classificar esses métodos segundo o elemento básico que a estrutura armazena: *pontos, linhas, retângulos* e *regiões* [Med95].

Como a dissertação faz uso de métodos de acesso já existentes, apresenta-se, a seguir, uma breve revisão dos mesmos.

Os principais métodos de acesso a pontos são o *Grid File*[NHS84], que divide o espaço numa grade genérica e está descrito mais adiante; a *Point Quadtree*[Ben75], que divide o espaço recursivamente em quadrantes; a *KD-tree* [BER85], que divide o espaço por hiperplanos ortogonais aos eixos e a *hB-tree* [ES93, ELS95] também descrita mais adiante. Exemplos de métodos de acesso a retângulos são a *SKD-tree*[OMSD87], uma variante da *KD-tree*; a *R-tree*[Gut84], que define uma árvore balanceada a partir de retângulos; e suas variantes: a R^+*-tree*[SRF87], uma árvore R que não permite sobreposição de nós; a R^**-tree*[BKSS90], uma árvore R cujos procedimentos de inserção e remoção de dados são otimizados; e a *PACKED R-tree*[RL85], que procura minimizar a fragmentação de memória dentro dos nós da árvore. A *PMR-quadtree*[Sam90] é um método de acesso a linhas que subdivide o espaço em quadrantes como as *Point Quadtrees* e a *VR-tree*[MCD94] armazena conjuntos de linhas longas. A *TR-tree*[SK91] armazena regiões, subdividindo-as em vários trapézios armazenados por uma árvore R.

Esta seção descreve com mais detalhes algumas dessas estruturas que servem de base para os métodos propostos no capítulo 4. A primeira delas é o *grid file* que armazena pontos, particionando o espaço a ser representado numa grade irregular genérica. Em seguida, descreve-se a estrutura das árvores R, usadas como métodos de acesso a retângulos (em geral o retângulo envolvente dos objetos propriamente ditos). Finalmente, descreve-se as *holey-brick trees (hB-trees)*, uma estrutura de acesso multi-indexado mais sofisticada também usada para o armazenamento de pontos.

Um *grid file* divide o espaço em células a partir de uma grade irregular. Essa divisão é feita por linhas ortogonais aos eixos ao longo de todo o espaço e armazenadas em *escalas* (um vetor para cada direção). Um *diretório* é uma matriz cujas entradas apontam para *buckets*. Cada célula do espaço corresponde a uma entrada do diretório e seus dados são armazenados no *bucket* indicado pelo diretório. Mais de uma célula pode estar associada ao mesmo *bucket*, como mostra a figura 2.4.

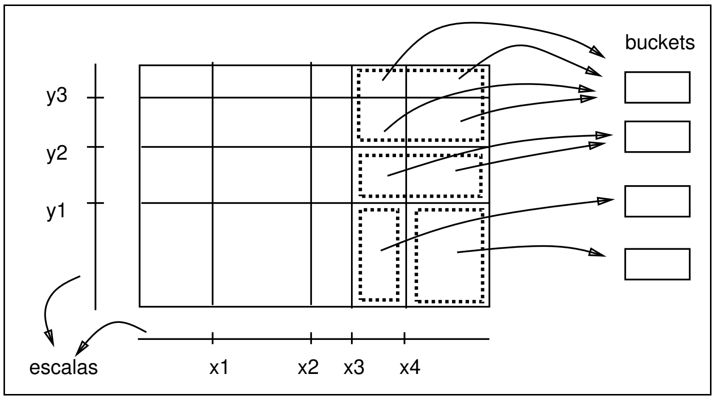

Figura 2.4: Estrutura de um *Grid File*.

As escalas são estruturas relativamente pequenas e podem ser mantidas na memória principal, mas um diretório normalmente é armazenado num conjunto específico de páginas em disco. Essa estrutura foi originalmente proposta em [NHS84] como método de acesso a arquivos multi-indexados e deixa em aberto a escolha das políticas de divisão e reunião de células e a implementação do diretório.

Uma árvore R (*R-tree*) é uma árvore balanceada, semelhante a *B-tree* na qual todas as folhas estão no mesmo nível (a uma mesma distância da raiz). Além disso, uma árvore R satisfaz os seguintes requisitos [BKS93, Med95]:

- a raiz possui pelo menos dois nós filhos;
- os nós intermediários contêm entre m e M entradas, onde m é o número mínimo de entradas num nó e M, o tamanho máximo, ambos parâmetros definidos previamente na definição da estrutura ($2 \leq m \leq M/2$);
- cada nó intermediário define um retângulo no espaço correspondente ao mínimo retângulo envolvente que contém os retângulos envolventes de seus filhos;
- a inserção de uma nova entrada em um nó com ocupação máxima provoca a

divisão do mesmo em dois. Essa nova entrada é inserida no nó pai e possíveis divisões são propagadas até a raiz, mantendo o balanceamento da estrutura.

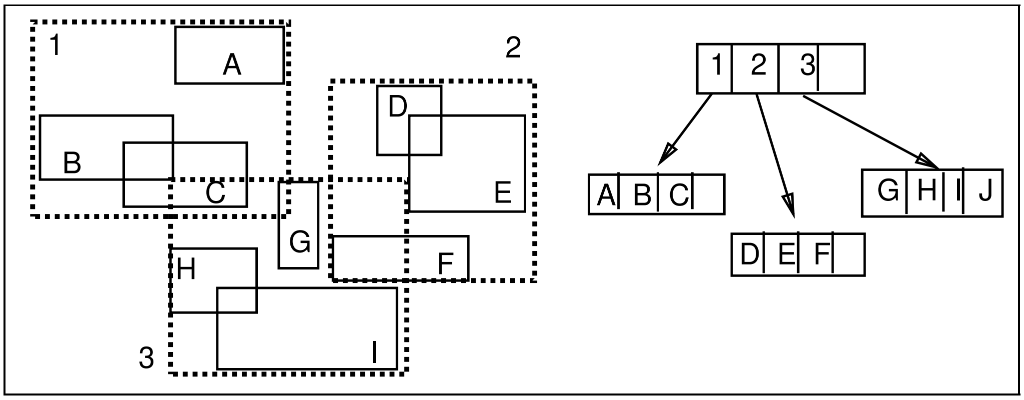

Figura 2.5: Um conjunto de retângulos representados em uma árvore R.

Um exemplo de árvore R é mostrada na figura 2.5. Como está definida acima, a árvore R permite sobreposição de nós, fazendo com que o espaço de busca possível a um objeto englobe mais de uma sub-árvore, comprometendo o desempenho da estrutura. A R^+*-tree* é uma variante da árvore R que não permite a sobreposição de nós, mas que possui algoritmos de edição mais complexos. Além disso, os retângulos inseridos muitas vezes têm de ser divididos para evitar a sobreposição dos nós, multiplicando o tamanho da árvore e piorando o desempenho do acesso aos dados [Med95]. A árvore R* [BKSS90] é uma outra extensão da árvore que visa otimizar seu desempenho através de novos algoritmos para a divisão dos nós, da reinserção de entradas durante a remoção de objetos afim de refinar a estrutura e da diminuição do número mínimo de entradas do nó para 40% do seu tamanho, melhorando a escolha na divisão dos nós.

Uma *hB-tree* [ES93] é uma árvore cujos nós representam blocos (retângulos multidimensionais, *bricks*) ou blocos furados (*holey bricks*), blocos de onde foram retirados blocos menores.

Um nó em uma *hB-tree* armazena índices e dados de forma uniforme usando *kd-trees*. Os nós de índice definem sub-espaços e os nós de dados armazenam os dados realmente. Se o sub-espaço descrito por um nó de dados possuir um buraco, ele descreve suas fronteiras internas atravé de uma *kd-tree*.

Uma *kd-tree* é uma árvore binária capaz de descrever uma região (possívelmente um buraco) do espaço. Uma folha dessa árvore pode (1) apontar para outros nós da *hB-tree*; ou (2) definir marcadores indicando que a sub-árvore é vazia (isso acontece quando um nó é dividido e parte do espaço passa a ser descrito por outro nó; ou (3) apontar para a representação dos dados própriamente dita (em nós de dados). Um exemplo de *hB-tree* pode ser visto na figura 2.6 (b) representando a região da figura 2.6 (a).

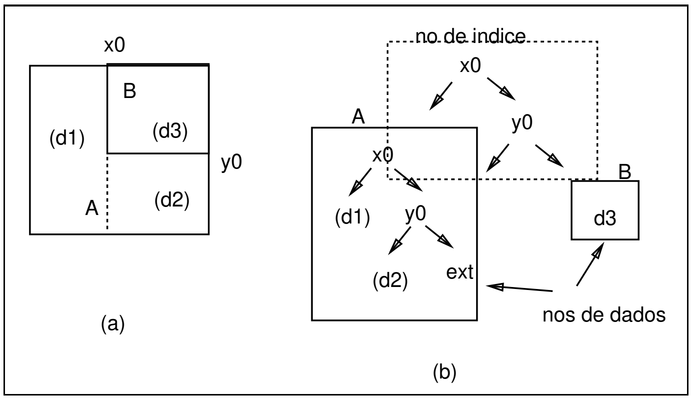

Figura 2.6: Exemplo de *hB-tree*.

2.4.2 Estruturas para Armazenamento Paralelo

Existem alguns trabalhos desenvolvidos com o intuito de melhorar o desempenho do acesso aos dados através da definição de estruturas de armazenamento paralelo. Diferente da arquitetura definida na dissertação, em geral a abordagem proposta utiliza uma arquitetura monoprocessada com múltiplos discos.

Em [KF92] e [FK93], são analisadas três estruturas baseadas na paralelização de árvores R num ambiente com um processador e múltiplos discos.

A primeira delas (*Independent R-tree*) distribui os retângulos a serem armazenados pelos diversos discos a partir da distribuição dos dados (por *round robing* ou *hashing*) ou por particionamento do espaço. Em cada disco, os retângulos são armazenados em uma árvore R própria. Apesar de não ter atingido os objetivos do artigo, esta estrutura parece ser a de mais fácil adaptação para o ambiente multi-processado, já que cada processador teria completo controle sobre a estrutura que armazena.

Em seguida, é analisada uma estrutura com *Super Nodes*, cada qual com tamanho de d páginas, onde d é o número de discos disponíveis. Os nós são distribuídos por todos os discos, uma página em cada. Essa estrutura espalha todos os objetos pelo sistema e mesmo numa consulta pequena todos os discos são envolvidos.

Finalmente, a última estrutura proposta, chamada *Multiplexed R-tree (MX-R-Tree)* define uma única árvore R, distribuindo seus nós pelo sistema de forma multiplexada. Os ponteiros para nós filhos incluem a identificação do disco onde é armazenado. Novamente, essa estrutura não se adequa à arquitetura proposta na dissertação, pois há forte acoplamento entre as diversas unidades do sistema.

Os artigos ainda propõem algumas heurísticas para a obtenção de uma distribuição uniforme dos dados. O balanceamento por dados (cada disco possui mesmo número de nós da estrutura) garante a distribuição uniforme dos dados. O balanceamento por área (cada disco cobre área de tamanho semelhante) é interessante, pois a chance de uma dada região se encontrar em cada um dos discos é a mesma. Porém, para mapas cujos dados não são distribuídos de forma uniforme pelo espaço, o balanceamento de carga será prejudicado. O critério de proximidade (nós próximos ou que se interceptam em discos diferentes para aumentar paralelismo) não se adequa ao problema da dissertação, pois aumentaria o grau de comunicação entre processadores na execução de operações.

A referência [Cia94] analisa as estruturas acima e propõe a partir delas estruturas para armazenamento paralelo baseadas em *grid files*. Na verdade, esse artigo faz duas contribuições interessantes: (1) introduz um método de distribuição baseado numa função de balanceamento (MC^2) que associa regiões do espaço criadas por divisão dinamica a discos; e (2) define os *Parallel Independent Grid File - PI-GF*, que visam o armazenamento dos dados em *grid files* distribuídos permitindo realocação dinâmica e visibilidade (a todo momento é possível saber a localização de determinada região). Mesmo propondo uma estrutura com diretórios dos *grid files* distribuídos, o artigo pressupõe que o acesso aos dados é feito por um único processador e que a distribuição dos dados objetive minimizar o tempo de acesso aos discos.

A principal diferença entre a linha de raciocínio destes trabalhos e o desenvolvido nessa dissertação dá-se no objetivo inicial escolhido. As estruturas comentadas acima se propõem a melhorar o desempenho do acesso aos dados, considerando o acesso a memória secundária como gargalo do problema. Como será discutido mais adiante, a dissertação parte da idéia de que o gargalo de desempenho num SIG está também na complexidade das operações espaciais, tornando interessante um ambiente de multi-processadores. Neste caso, é importante que haja uma certa independência entre os processadores, resultando numa arquitetura fracamente acoplada, de forma a paralelizar a execução de operações sobre mapas.

2.4.3 Algoritmos de Manipulação de Mapas

Os métodos de armazenamento propostos na dissertação pretendem constituir uma arquitetura eficiente para a síntese de operações geográficas. Alguns trabalhos que tratam deste assunto estão discutidos a seguir.

[RS94] apresenta uma definição matemática para mapas e sua topologia e define procedimentos para o cálculo da superposição entre mapas, a partir de algoritmos de varredura.

[WH92] propõe um algoritmo para o cálculo de superposição de mapas num ambiente paralelo. A estrutura de dados usada, porém, é centralizada e não faz uso de indexação espacial (conjunto de linhas). A distribuição de dados durante a execução é definida a partir de uma grade regular.

[Wan93] propõe e compara algoritmos paralelos para *map overlay*, a partir do cálculo da interseção de linhas. Novamente, não faz uso de indexação espacial nem do armazenamento distribuído.

Nenhum dos artigos citados até aqui fazem uso de indexação espacial ou de métodos de acesso espacial em geral.

[Sch90] e [WDYT91] propõem e analisam diferentes algoritmos para junção paralela (*parallel sort merge join, parallel hash join*) usados em bancos de dados paralelos tradicionais. Porém, estes algoritmos não se adequam a dados espaciais [BKS93].

[BKS93] descreve um algoritmo eficiente para junções espaciais usando métodos de acesso pré-existentes. Ele parte da idéia de explorar os índices espaciais inerentes ao método de acesso na execução de junções. O processamento de consultas espaciais é executado em dois passos: filtro e refinamento. O artigo analisa o uso de dois tipos de métodos de acesso: (1) divisão em grade regular (*quadtree*) e (2) divisão uniforme dos dados (*R-tree*). No primeiro caso, o filtro tem alto custo (mesmo objeto é replicado em várias células), mas poucos objetos participam da fase de refinamento.

A vantagem ressaltada no segundo caso é o fato da árvore ser balanceada e objetos estarem representados por seus retângulos envolventes. A desvantagem das árvores R é não garantir a decomposição disjunta dos retângulos.

O objetivo desse artigo é próximo ao da dissertação, porém não considera paralelismo nem particionamento dos dados. A desvantagem no uso de grades pode ser resolvida com a introdução de fragmentação dos objetos ao invés de sua duplicação. O fato de células da grade garantirem a presença dos objetos situados numa mesma região do espaço se encontrarem no mesmo processador passa a ser uma vantagem sobre estruturas do outro tipo. As árvores R, num ambiente paralelo, têm o problema de superposição de nós minimizado no acesso a dados (sub-árvores distintas são percorridas por processadores diferentes), mas podem aumentar quando na execução de operações entre conjuntos distintos que exigirão maior comunicação entre processadores.

[BKSS94] refina o algoritmo estudado no artigo anterior definindo três procedimentos para junção espacial. O primeiro faz uso do método de acesso, filtrando o domínio de resposta pela interseção dos retângulos envolventes. O segundo filtro identifica pares de objetos que se interceptam sem que seja necessário checar sua geometria exata. Finalmente, o terceiro passo testa o restante dos pares, comparando sua geometria exata por dois métodos: varredura do plano e *TR-trees* que decompõem os polígonos em trapezóides.

Como será explicado no capítulo 5, o esforço na definição de algoritmos sobre o método de armazenamento proposto é complementar aos discutidos acima, já que faz uso deles para o processamento de operações locais.

[HS94] agrupa as duas visões: uso do método de acesso espacial e paralelização de operações complexas que requerem uso intensivo de CPU. O artigo propõe algoritmos para a criação da estrutura de dados (que pode ser feita dinamicamente), a poligonização e junção espacial (*buffer* de linhas). São analisados três métodos de acesso: (1) a árvore R que apresenta como desvantagem a superposição de nós; (2) a árvore R^+, na qual a decomposição ainda é dependente dos dados, dificultando operações entre conjuntos; e (3) a *bucket-PMR-quadtree* que tem como vantagem a decomposição regular do espaço.

A decomposição não disjunta do espaço é de fato uma desvantagem (apesar do acesso concorrente a diferentes sub-árvores) que deve ser tratada de forma especial na execução de junções espaciais. A dependência dos dados em sua partição não prejudica o desempenho dos algoritmos de junção, se estes levam a indexação espacial em consideração. Novamente, os objetivos desse artigo diferem dos da dissertação, pois ele considera o armazenamento dos dados centralizado não levando em consideração o gargalo de acesso a memória secundária.

Capítulo 3

Gerenciadores de Objetos Geográficos

A seção 3.1 descreve, em linhas gerais, o modelo para dados geográficos considerado, seguido da formalização dos conceitos relativos a mapas vetoriais, seus elementos componentes e sua fragmentação.

A seção 3.2 descreve uma arquitetura genérica para gerenciadores de objetos geográficos (GOG) que integra três sub-sistemas de armazenamento e manipulação, um para cada tipo de representação (matricial, vetorial e convencional). Em seguida, são propostas extensões da arquitetura para um ambiente paralelo.

Finalmente, a seção 3.3 reduz o problema abordado ao módulo vetorial da arquitetura e descreve uma proposta para a organização de mapas no banco de dados.

3.1 Modelos de Dados

O objetivo desta seção é descrever o contexto onde as estruturas de dados propostas neste trabalho se inserem. Inicialmente, o texto apresenta o modelo de dados para objetos geográficos considerado. Em seguida, descreve-se formalmente os conceitos relativos à classe de objetos tratado neste trabalho: mapas vetoriais, seus elementos componentes e sua fragmentação.

3.1.1 Um Modelo para Objetos Geográficos

Esta seção descreve brevemente o modelo para objetos geográficos adotado neste trabalho, seguindo em linhas gerais [CFS+94]. O modelo introduz os conceitos de geo-objeto, geo-campo, mapa de geo-objetos e representação de geo-campo.

Segundo [Gut94], duas visões alternativas podem ser consideradas ao representar informações geográficas em um SIG. A primeira delas visa representar as entidades presentes no espaço e a outra pretende representar o próprio espaço, associando informações a todos os seus pontos.

Um *geo-objeto* corresponde a um elemento da realidade geográfica e tipicamente possui atributos *convencionais*, além de estar associado a uma ou mais *representações* geométricas geo-referenciadas. Essas representações são descrições da sua localização na superfície terrestre. Por exemplo, uma estrada é um geo-objeto com atributos convencionais indicando sua identificação, tipo, etc... e pode ser representada por uma linha em um mapa.

As representações de um geo-objeto podem ser tratadas como componentes de outros objetos, chamados de *mapas de geo-objetos*, conceito definido em detalhes na seção 3.1.2. Um mapa de geo-objetos é constituído de um conjunto de elementos geométricos, cada um deles indicando forma e localização no espaço de geo-objetos a eles associados. Um exemplo típico pode ser visto na figura 3.1 que representa a hidrografia de uma região, onde as linhas r1 e r3 representam dois rios, a região r2, um rio e a região l1, um lago. Um mapa de geo-objetos pode, também, ter atributos convencionais, como escala, projeção cartográfica, etc...

Um *geo-campo* define a distribuição espacial de uma variável geográfica sobre uma região da superfície terrestre, como altitude ou tipo de solo. Quando a variável assume um conjunto finito de valores, o geo-campo normalmente é chamado de *campo temático*. Um geo-campo pode também ter atributos convencionais, como a data em que foi definido ou o órgão responsável pelo levantamento, e estar associado a uma ou mais *representações* da distribuição da variável geográfica.

Novamente, as representações de um geo-campo podem ser tratadas como objetos separados de tipos especiais (ver [CFS+94]). Por exemplo, a altimetria de uma região pode ser representada por curvas de nível ou por uma imagem, onde a cor indica altitude. Em particular, um campo temático pode ser representado por um *mapa*

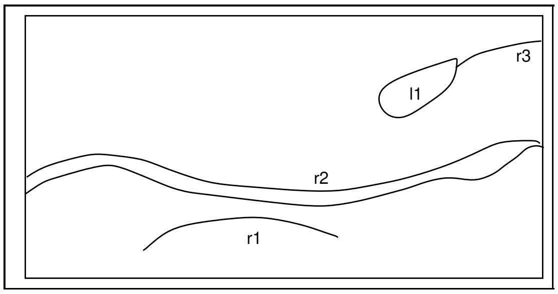

Figura 3.1: Hidrografia de uma região representada num mapa de geo-objetos.

temático, conceito também definido em detalhe na seção 3.1.2.

Quando o contexto permitir, chamaremos mapas de geo-objetos e representações de geo-campos genericamente de *representações*.

Considera-se que há um relacionamento um-para-muitos entre um geo-campo e suas representações e um relacionamento muitos-para-muitos entre geo-objetos e mapas de geo-objetos (onde os geo-objetos possuem uma representação). Um *banco de dados geográfico* é, então, uma coleção de geo-objetos, geo-campos e representações e uma coleção de relacionamentos entre estes objetos.

Um banco de dados geográfico pode, em princípio, ter uma implementação distribuída a partir das mesmas técnicas usadas em um banco de dados convencional. Porém, implementações possivelmente mais eficientes poderão ser obtidas levando em conta duas características importantes de bancos de dados geográficos: (1) o armazenamento das representações tipicamente ocupa muito mais espaço do que o ocupado pelos atributos convencionais dos objetos; (2) as operações espaciais sobre estas representações são, em geral, muito mais complexas e dispendiosas do que operações envolvendo apenas os atributos convencionais.

3.1.2 Mapas Temáticos e Cadastrais

Apesar de muitas vezes ocuparem espaço de armazenamento menor que, por exemplo, objetos representados de forma matricial, objetos vetoriais são normalmente caracterizados por geometria complexa, pois se propõe a representar objetos geográficos de uma forma precisa em termos de posição, forma, dimensão e tamanho.

Dada sua importância para sistemas de geoprocessamento, esta seção define em detalhe o conceito de mapa vetorial, entendido como base para representações de mapas de geo-objetos e mapas temáticos. A descrição a seguir toma como base [PD95] e define, ainda, critérios para fragmentação de mapas vetoriais. Contém, também, breves considerações acerca de operações sobre mapas vetoriais.

Um *ponto* é um par ordenado $(x, y) \in \Re^2$.

Uma *linha* é uma função contínua $l : I \rightarrow \Re^2$, onde $I = [0, 1]$.

Uma linha é *fechada* se e somente se $l(0) = l(1)$; caso contrário é *aberta*. Os pontos $l(0)$ e $l(1)$ são chamados *nós* ou *fronteira* de l.

Uma linha l é *simples* se e somente se l define uma função injetora em I, se l for aberta, ou se l define uma função injetora em $[0, 1)$, se l for fechada. Intuitivamente, l é simples se não cruza a si mesma em algum ponto.

Dada uma linha l e $a, b \in [0, 1]$, o *segmento de linha* de l entre a e b, denotado por $l_{a,b}$, é a função l restrita ao intervalo $[a, b]$.

Uma *cadeia* é uma seqüência de linhas $c = (l_0 ... l_{k-1})$ tal que, para todo $i \in [1, k-1]$, temos $l_{i-1}(1) = l_i(0)$. Note que, a partir de c, podemos definir uma linha l_c, chamada de linha *induzida* por c. Uma cadeia c é *simples*, *aberta* ou *fechada* se e somente se l_c o for.

Uma *região* é um par $r = (c_0, F)$, onde c_0 é uma cadeia fechada simples, chamada *fronteira externa* da região, e F é um conjunto $r_1, ..., r_n$, possivelmente vazio, de regiões, onde $n \geq 0$. As fronteiras externas de r_i são chamadas de *fronteiras internas* da região r e são tais que não se interceptam [1], estão contidas no interior[1] de c_0 e, para todo $i, j \in [1, n]$ com $i \neq j$, temos que r_i está no exterior[1] de r_j. Intuitivamente, as fronteiras internas $r_1, ..., r_n$ definem buracos na região r. De fato, a definição recursiva acima possibilita a existência de buracos dentro de buracos, ou seja, a representação de uma ilha dentro de um lago no interior de um território, por exemplo.

Um *elemento geométrico* é um ponto, uma linha ou uma região. Pontos, linhas e regiões possuem *grau* 0, 1 e 2, respectivamente.

Define-se, ainda, um conjunto de relações entre pares de elementos geométricos. As

[1]As relações topológicas são definidas mais adiante.

relações topológicas usadas nesse texto estão definidas em [EF91]. A seguir, apresenta-se de forma intuitiva um pequeno resumo das definições das relações básicas mais usadas [CFO93, Toc93, Gut94]. Essas relações são disjuntas entre si, ou seja, se um par de elementos satisfaz uma das relações, ele não satisfaz nenhuma das outras. São elas: adjacência, inclusão, cruzamento, interseção, cobertura, disjunção e igualdade. Intuitivamente, *adjacência* identifica elementos que se tocam; *inclusão* indica elementos no interior da fronteira de outro elemento de mesmo ou maior grau; *cruzamento* indica que um elemento atravessa a fronteira de outro; *interseção* indica que uma parte de dois elementos de mesmo grau é comum a ambos; *cobertura* é definido de forma que um elemento contém outro *e* parte de suas fronteiras é comum a ambos; *igualdade* indica que os dois elementos se localizam no mesmo espaço geométrico e tem a mesma forma. Finalmente, *disjunção* indica que o par de elementos não satisfaz nenhuma das condições anteriores. A figura 3.2 ilustra exemplos dessas relações.

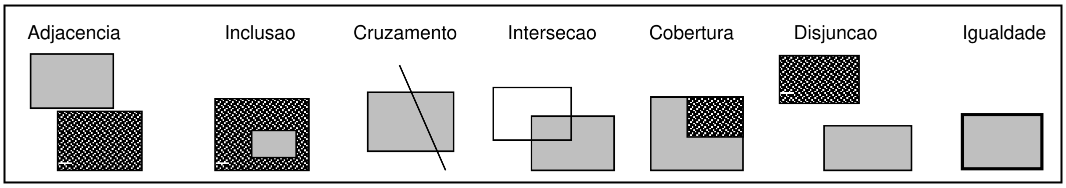

Figura 3.2: Exemplos de cada uma das relações topológicas definidas.

Um *mapa vetorial*, ou simplesmente um *mapa*, é uma tripla $M = (P, L, R)$, onde P, L e R são respectivamente conjuntos de pontos, linhas e regiões, os *elementos* do mapa, tais que:

1. dadas duas regiões r e $s \in R$, então:
   - r e s são disjuntas; ou
   - r e s se tocam (em uma linha ou ponto), ou seja, são adjacentes;
2. dadas uma linha $l \in L$ e uma região $r \in R$, então:
   - l e r são disjuntas; ou
   - l e r se tocam em um ponto p, que deve ser um nó de l e pertencer à fronteira de r;
   - l faz parte da fronteira de r; ou
   - l é interior a r; ou

- l é coberta por r (ou seja, l é interior a r, mas pelo menos um nó de l faz parte da fronteira de r);

3. dados um ponto $p \in P$ e uma região $r \in R$, então:
   - p e r são disjuntos; ou
   - p está na fronteira de r (ou seja p é nó de duas linhas consecutivas da fronteira de r); ou
   - p é interior a r;
4. dadas duas linhas l e $m \in L$, então:
   - l e m são disjuntas; ou
   - l e m se tocam em um nó comum;
5. dados um ponto $p \in P$ e uma linha $l \in L$, então:
   - p e l são disjuntos; ou
   - p é nó de l.

Um *mapa temático* é uma tripla $m = (C, M, f)$ onde C é um conjunto finito de *temas*[2], $M = (P, L, R)$ é um mapa vetorial e $f : P \cup L \cup R \rightarrow C$ é uma função que associa cada elemento de M a um tema em C. Um *mapa de geo-objetos* é definido de forma idêntica, exceto que C é um conjunto finito de geo-objetos. Dado um geo-objeto $c \in C$, diz-se que $f^{-1}(c)$ é a *representação* de c em m.

As estruturas de dados descritas no capítulo 4 implementam genericamente a noção de mapa vetorial. Características que distinguem mapas temáticos e de geo-objetos são implementadas em rotinas específicas de edição. Deste modo, o texto usará a denominação genérica mapa ao referir-se a mapa vetorial, salvo alguma consideração em contrário.

A figura 3.3 exemplifica dois mapas de geo-objetos desenhados em conjunto. O primeiro deles é o mesmo da figura 3.1 e representa a hidrografia de uma região. O segundo mapa representa a localização espacial de objetos geográficos, onde as regiões f1 e f2 representam duas fazendas, as linhas e1 e e2, duas estradas e os pontos c1 e c2, duas casas.

[2]Um tema é um valor possível da variável geográfica representada - ver seção 3.1.1

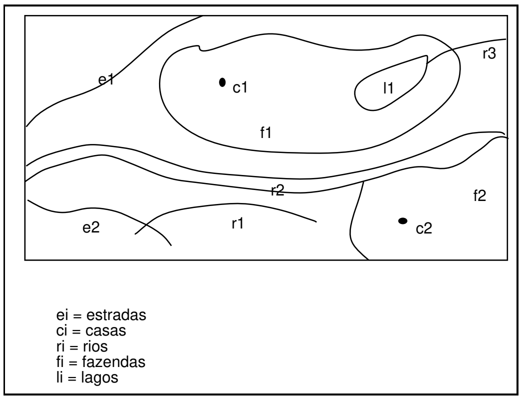

Figura 3.3: Exemplo de mapas.

Dado um mapa vetorial $M = (P, L, R)$ e uma região u, o *fragmento de M induzido por u* é o mapa $M' = (P', L', R')$ definido da seguinte forma:

- P' é o subconjunto dos pontos em P tal que $p \in P'$ se e somente se $p \in P$ e p está:
  - no interior de u;
  - ou na fronteira de u;
- $l' \in L'$ se e somente se existe $l \in L$ tal que
  - l cruza u e l' corresponde ao fragmento de l que está no interior de u; l' é chamado de *elemento fragmentado* de M'.
  - ou l está no interior de u e $l = l'$.
- $r' \in R'$ se e somente se existe $r \in R$ tal que
  - r e u se interceptam e r' corresponde ao fragmento de r que está no interior de u; r' é chamado de *elemento fragmentado* de M'.
  - ou r está no interior de u e $r = r'$.

Um *critério de fragmentação de mapas* para uma região u de $\Re^2$ é um conjunto finito $F = \{f_1, ..., f_n\}$ de regiões que cobrem u, chamadas *células de fragmentação* ou *células*. Dado um conjunto μ de mapas, F induz, então, um segundo conjunto μ' consistindo dos fragmentos dos mapas em μ induzidos pelas regiões em F. As fronteiras das regiões que induzem a fragmentação de mapas definem *fronteiras artificiais* no mapa.

Define-se, ainda, um operador de *desfragmentação* de mapas $\cup_{\oplus}$ que une os fragmentos de uma mapa, retirando fronteiras artificiais e recompondo os elementos fragmentados. Dado um conjunto $\nu = M_0, M_1, ...$ de fragmentos de um mapa M, tais que $M_i = (P_i, L_i, R_i)$, o operador $\cup_{\oplus}(\nu)$ recompõe o mapa $M = (P, L, R)$ tal que:

- P é a união dos conjuntos P_i.
- L é a união modificada dos conjutos L_i, tal que $l \in L$ se e somente se $l \in L_i$ ou $l = l_i \oplus$[3] l_j e $l_i \in L_i$ e $l_j \in L_j$.
- R é a união modificada dos conjutos R_i, tal que $r \in R$ se e somente se $r \in R_i$ ou $r = r_i \oplus r_j$ e $r_i \in R_i$ e $r_j \in R_j$.

Por fim, consideramos como uma operação sobre mapas qualquer função que tenha como argumento ou contra-argumento um conjunto de mapas. Um exemplo importante são as chamadas *consultas por janela* (*window queries*) para mapas, que recebem como entrada um mapa M e uma região u (normalmente definida pelas extremidades de um retângulo) e retorna o fragmento de M induzido por u, conforme definido acima. Outro exemplo importante são as operações de junção espacial para mapas, que possuem como argumento dois mapas M e M' e geram como resposta um terceiro mapa M'' (ou um valor numérico v), onde M'' (ou v) é construído utilizando-se elementos dos mapas de entrada que possuem entre si um dado relacionamento topológico ou geométrico (por exemplo interseção, cruzamento, pertinência ou proximidade). Operações sobre mapas serão exploradas com mais detalhes no capítulo 5.

3.2 Arquiteturas para Gerenciadores de Objetos Geográficos

Uma arquitetura define a organização dos dados. Esta seção propõe uma arquitetura para um gerenciador de objetos geográficos e suas extensões num ambiente distribuído. Em seguida, limita-se o escopo do trabalho, definindo o módulo vetorial da

[3] Observação: $\oplus$ é um operador que compõe partes de um elemento fragmentado.

arquitetura, responsável pelo armazenamento e manipulação dos objetos definidos em 3.1.2.

3.2.1 Uma Arquitetura Genérica para um Gerenciador de Objetos Geográficos

Esta seção descreve brevemente uma arquitetura em camadas para um Gerenciador de Objetos Geográficos, ou brevemente GOG, calcada nos conceitos introduzidos na seção 3.1.1. Em linhas gerais, um GOG possui facilidades para o armazenamento, manipulação e vizualização de objetos geográficos. Uma descrição mais detalhada, incluindo a definição dos dados e funções definidas em cada sub-nível, pode ser encontrada em [HMS+94].

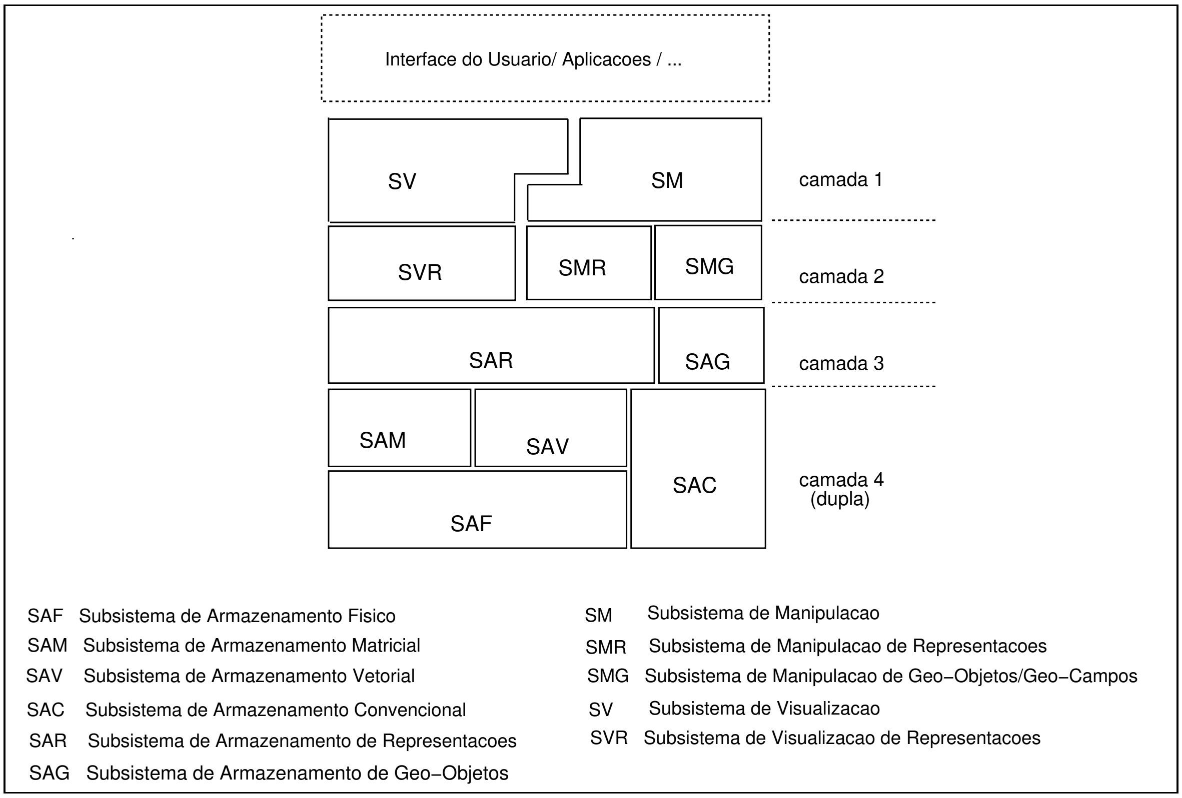

Figura 3.4: Arquitetura em camadas para um Gerenciador de Objetos Geográficos.

A figura 3.4 indica as camadas e sub-sistemas propostos para organizar um GOG. No nível superior, não especificado pela arquitetura, encontram-se aplicações desenvolvidas e módulos de interface com o usuário que pode ou não ter acesso direto à funcionalidade oferecida pelos Sub-sistemas de Visualização (SV) e de Manipulação (SM).

A primeira camada separa claramente as questões de visualização dos problemas de manipulação e oferece as abstrações de geo-objetos, geo-campos, e representações em geral (ou seja, relembrando, mapas de geo-objetos e representações de geo-campos). O *Sub-sistema de Visualização* (SV) oferece funções básicas para visualização de objetos, tipicamente mais complexas do que aquelas oferecidas para dados tradicionais. O *Sub-sistema de Manipulação* (SM) oferece rotinas para definição e manipulação de geo-objetos, geo-campos e representações, indistintamente, de forma mais próxima ao modelo de dados. Um dos objetivos deste sub-sistema é mascarar a dualidade existente nos sistemas de armazenamento que tratam separadamente os diversos tipos de formatos de dados. Além disso, ele oferece funções de definição e gerência de bancos de dados; processamento de consultas, de atualizações e de operações; e controle de concorrência e integridade.

A segunda camada desce um nível na escala de abstração, separando a parte convencional da parte espacial. Ou seja, ela oferece separadamente serviços de visualização e manipulação de alto nível para representações, através do *Sub-sistema de Visualização de Representações* (SVR) e do *Sub-sistema de Manipulação de Representações* (SMR), respectivamente, e serviços de manipulação dos atributos convencionais dos objetos, através do *Sub-sistema de Manipulação de Geo-objetos* (SMG). Esses sub-sistemas disponibilizam operações usuais de banco de dados, como a recuperação e junção por atributos, cada qual especializado no tratamento de dados específicos.

A terceira camada cobre serviços de armazenamento e manipulação elementar para representações, via o *Sub-sistema de Armazenamento de Representações* (SAR), e para atributos convencionais, através do *Sub-sistema de Armazenamento de Geo-objetos* (SAG). Esses sub-sistemas são responsáveis pelo armazenamento persistente e edição elementar desses dados.

A última camada (dupla) implementa as formas de armazenamento - matricial, vetorial ou convencional - através respectivamente do *Sub-sistema de Armazenamento Matricial* (SAM), *Sub-sistema de Armazenamento Vetorial* (SAV) e *Sub-sistema de Armazenamento Convencional* (SAC). Os dois primeiros subsistemas fazem uso do serviço de armazenamento de páginas físicas disponibilizada pelo *Sub-sistema de Armazenamento Físico* (SAF). O SAV trabalha com abstrações de *ponto*, *linha*, *polígono* e *região* implementadas através de estruturas espaciais, como por exemplo R-trees, V-trees e VR-trees [Med95].

A arquitetura para um GOG emprega uma abordagem híbrida, usando diferentes sistemas de armazenamento para cada tipo de objeto, mas os trata de forma uniforme na interface com a aplicação a nível conceitual. Essa política permite que sejam usados um Sistema Gerenciador de Banco de Dados convencional para o armazenamento de atributos não espaciais e métodos de acesso específicos para o armazenamento de representações espaciais. Neste contexto, a camada superior da arquitetura tem como objetivo integrar os diferentes sub-sistemas da arquitetura, tornando a divisão transparente para o usuário final e suas aplicações.

Outra vantagem desta arquitetura modular está na possibilidade de tratá-los separadamente, resolvendo problemas específicos de cada tipo de dado sem se preocupar com os demais. A seção 3.3 faz uso desta idéia para restringir o problema tratado na dissertação ao módulo vetorial da arquitetura.

A seguir, discute-se como adaptar a arquitetura para um ambiente distribuído.

3.2.2 Arquiteturas Paralelas para Gerenciadores de Objetos Geográficos

Como foi discutido no capítulo 1, com o advento do uso de arquiteturas paralelas é razoável que se estude seu uso para incrementar o desempenho de gerenciadores de objetos geográficos. Como vantagem direta do uso de um ambiente distribuído, obtém-se uma maior capacidade de armazenamento sem que haja perda de desempenho, já que múltiplos processadores cooperam para a resolução das operações. Novos problemas, porém, surgem como, por exemplo: complexidade na administração dos dados; considerações na distribuição de dados; problemas de balanceamento de carga, etc...

Uma das formas de paralelizar as operações em um banco de dados, chamada de paralelismo particionado [DG92], consiste em particionar os dados e distribuí-los entre vários processadores. Na arquitetura clássica para sistemas de gerência de bancos de dados distribuídos, o armazenamento dos dados nos vários processadores é guiado por um *critério de distribuição*, que pode ou não incluir replicação de fragmentos dos dados. O ciclo de execução de uma consulta ou atualização passa a incluir, então, execução de operações locais, transferências de dados e composição de resultados parciais.

A adaptação desta idéia genérica à arquitetura em camadas descrita na seção anterior origina quatro possíveis arquiteturas distribuídas, dependendo do nível em que se introduz o critério de distribuição dos dados. A fronteira que as diferencia é sutil, variando o grau de abstração dos dados fragmentados e das operações paralelizáveis. A camada onde é aplicado o critério passa a ser responsável por mascarar para a camada superior a distribuição do armazenamento e manipulação dos dados, controlando a localização dos fragmentos dos dados, bem como o processamento distribuído das operações oferecidas naquela camada.

Considera-se inicialmente que a distribuição se dá na primeira camada (ver figura 3.4). Neste caso, pode-se definir diversos critérios de distribuição para geo-objetos, geo-campos e representações, de forma semelhante aos usados em bancos de dados convencionais. Porém, uma forma de distribuição baseada na fragmentação das representações oferece maior flexibilidade para paralelizar operações espaciais, já que permite armazenar fragmentos de representações em processadores distintos. O corte da arquitetura nesta camada resulta em um banco de dados distribuído fracamente acoplado, onde cada processador conhece a semântica dos dados e implementa operações de alto nível, tornando-os bastante independentes entre si.

A distribuição ao nível da segunda camada facilita a definição de critérios de distribuição para as representações de forma completamente independente dos critérios para a parte convencional dos objetos. Porém, cada um dos processadores ainda conhece a semântica das representações e pode operar sobre elas de forma independente.

A distribuição dos dados na terceira camada (SAR e SAG) novamente é definida sobre representações e atributos convencionais, separadamente. Ela se distingue da situação anterior pelo nível de abstração dos operadores paralelizados, que é menor neste caso, pois tratam especificamente das estruturas de armazenamento e acesso aos dados. Esse é o último ponto de corte em que os diversos processadores conhecem a semântica das representações.

Estas arquiteturas podem ser bastante adequadas no contexto de um *cluster* de estações de trabalho, por exemplo, oferecendo um ambiente de armazenamento distribuído, interligado por um sistema de comunicação de alto desempenho.

O último ponto de corte possível é no nível da camada de armazenamento elementar. Neste caso, apenas operações básicas sobre a geometria dos objetos vetoriais (no SAV), sobre estruturas matriciais (no SAM) ou sobre atributos convencionais (no SAC) são paralelizadas. Os processadores que armazenam os fragmentos destes objetos são completamente dependentes dos processos que implementam as camadas superiores. Essa abordagem é adequada a arquiteturas cliente/servidor com serviços orientados a objetos elementares ou a páginas (corte no SAF).

Como já foi observado, a distinção entre a introdução do ponto de corte nos diferentes níveis da arquitetura é sutil. O trabalho desenvolvido pela dissertação pressupõe que os diferentes nós do sistema tenham conhecimento da semântica de mapa de forma a reutilizar com facilidade a definição de operações sobre mapas como operações sobre fragmentos de mapas (ver seção 3.1.2). Deste modo, supõe-se uma distribuição nos níveis 1, 2, ou 3 da arquitetura, dependendo do grau de abstração das operações onde é introduzido o critério de distribuição. Sugere-se que a distribuição seja feita a nível da camada 2, onde a separação dos atributos espaciais e não espaciais está bem definida e estão presentes operações tanto de armazenamento como de manipulação de objetos.

Sendo mais específico, pode-se considerar que os métodos de armazenamento propostos no capítulo 4 por si só implementam o critério de distribuição na camada 3 da arquitetura, pois os subsistemas desta camada somente tratam de operações básicas de armazenamento e edição das estruturas de dados propostas. Operações sobre dados armazenados desta forma, não necessariamente fazem uso das vantagens introduzidas pala distribuição do armazenamento da estrutura. Com a introdução da resolução paralela de operadores espaciais, no capítulo 5, passa-se a considerar realmente a distribuição da arquitetura na camada 2, num nível mais elevado de abstração.

3.3 Estratégias para o Armazenamento de Mapas Vetoriais

Limitando o escopo da arquitetura do gerenciador de objetos geográficos discutida até aqui ao tratamento de mapas vetoriais, a tabela 3.1 indica os subsistemas descritos na seção 3.2.1 que são responsáveis pelo tratamento de mapas e indica o nível de abstração dos objetos manipulados por cada um deles.

SM	geo-objetos, geo-campos, representações
SMR	mapas temáticos / mapas de geo-objetos
SAR	mapas temáticos / mapas de geo-objetos
SAV	pontos, linhas, polígonos, regiões, conjuntos
SAF	páginas

Tabela 3.1: Subsistemas e objetos manipulados

O conceito de mapa é nativo, então, das três camadas superiores do GOG. Isto significa que a decisão sobre em qual delas será introduzido o armazenamento distribuído de mapas deve levar em consideração apenas o grau de abstração das operações a serem paralelizadas.

A implementação de estruturas que suportem a arquitetura proposta exigem a adaptação do(s) sub-sistema(s) onde é feito o corte da estrutura que deve mapear operações centralizadas em um conjunto de operações locais a serem aplicadas em cada um dos nós do sistema.

Independentemente de qual for o nível em que se introduz distribuição, propõe-se trabalhar com critérios de distribuição para mapas definidos sempre a partir de critérios de fragmentação, ou seja, os processadores armazenarão localmente fragmentos de mapas. Esta forma de distribuição de mapas tem como conseqüência imediata que uma operação geográfica sobre um mapa será traduzida em operações geográficas semelhantes sobre seus fragmentos, de forma transparente para os usuários.

Discute-se, a seguir, possíveis formas de organização dos mapas vetoriais num banco de dados paralelo. Considera-se, para efeito de armazenamento de mapas, o uso das estruturas de dados usuais para conjunto de objetos espaciais, como árvores R, *quadtrees*, etc... A análise não leva em conta características específicas de um determinado método de acesso, possibilitando o uso de estruturas baseadas tanto na partição do espaço modelado, como em estruturas construídas a partir da distribuição efetiva dos dados.

O primeiro modelo possível (figura 3.5(a)) armazena cada um dos mapas do banco de dados numa estrutura de dados própria, distribuída sem levar em consideração as demais estruturas. Essa política faz com que o acesso a cada um dos mapas seja consideravelmente eficiente, já que faz uso de um método de acesso espacial com bom desempenho, melhorado através da arquitetura paralela que dá suporte ao armazenamento de grandes volumes de dados.

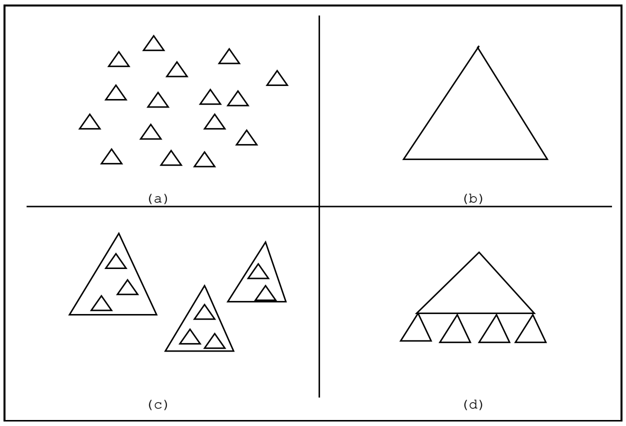

Figura 3.5: Políticas de organização de um banco de dados com método de acesso baseado em árvores paralelas

A dificuldade surge na execução de consultas que envolvem mais de um mapa com a necessidade de junção entre duas ou mais estruturas. Como a decomposição de cada mapa é feita de forma independente, assim como a distribuição das respectivas sub-estruturas pelo sistema paralelo, nada garante, no caso geral, que objetos que se relacionam se encontrem sob a responsabilidade do mesmo processador, logo o grau de comunicação entre os nós do sistema tende a ser muito elevado comprometendo o desempenho dessas operações mesmo num ambiente de rede extremamente rápido.

Uma alternativa oposta a essa política (figura 3.5(b)) é armazenar todo o banco de dados numa única estrutura de acesso. Desta forma, faz-se com que objetos próximos estejam juntos na estrutura (considera-se o uso de uma política de clusterização na construção da estrutura), independente dos mapas aos quais pertencem. Em outras palavras, privilegia-se o agrupamento espacial dos dados armazenados, sem considerar sua semântica.

Na verdade, objetos de diferentes mapas se encontram numa mesma sub-estrutura e portanto num mesmo processador, segundo sua localização. Assim, a troca de informações entre nós do sistema durante operações que envolvem mais de um mapa se restringem a possíveis processadores que cobrem áreas sobrepostas. A comunicação é bem menor do que na alternativa anterior, levando a um melhor desempenho desse tipo de operação.

Essa organização cria, porém, dois problemas. Em primeiro lugar, é necessária a extensão da definição básica do método de acesso usado de modo a criar uma forma de identificar o mapa ao qual determinado objeto pertence, pois os objetos de todos os mapas são armazenados numa única estrutura de armazenamento. Em segundo lugar, o acesso a objetos de um mapa passa a ter que considerar mais uma variável que identifica a qual um mapa cada objeto está associado, comprometendo de algum modo a eficiência da estrutura.

Independente disso, essa organização por si só tende a espalhar demais os objetos de um único mapa com o crescimento exagerado do tamanho da estrutura de acesso. Partindo apenas da observação do porte da estrutura a ser percorrida (em média, n vezes maior por armazenar n mapas) é possível perceber a perda considerável do desempenho do método de acesso.

Uma solução de compromisso (figura 3.5(c)) entre as duas políticas acima seria a criação de um banco de dados com um conjunto de super-estruturas, cada uma delas armazenando um conjunto de mapas definidos pelo DBA ou pelo modelador de aplicações do banco de dados, segundo o relacionamento semântico entre eles. Ou seja, é feito um agrupamento de mapas fortemente acoplados em estruturas de armazenamento comuns de forma a privilegiar consultas com operações entre eles. Essas estruturas, porém, não são exageradamente extensas, pois não englobam todo o banco de dados, diminuindo a perda observada no acesso à estrutura global proposta anteriormente.

Essa organização possui o mesmo *overhead* da anterior, já que ainda é necessário criar extensões à definição da estrutura para suportar a identificação de mapas inseridos numa única estrutura, tornando mais complexa a gerência do banco de dados. A grande diferença está no tamanho das estruturas, incrementando o desempenho das mesmas, já que cada super estrutura armazena um número pequeno de mapas normalmente acessados de forma integrada. Tenta-se, aqui, equilibrar o esforço necessário para a resolução de dois tipos de consulta, as que envolvem apenas acesso a um mapa e as que envolvem junções espaciais.

O fato de deixar sob a responsabilidade de um administrador de dados a definição da arquitetura do banco é razoável, pois conhecendo o perfil das aplicações, é possível fazer uma boa avaliação da necessidade de bom desempenho em determinadas consultas, por serem mais comuns ou mais extensas. A automação deste processo através de utilitários estatísticos usados pelo otimizador do SGBD pode ser também considerada.

Uma última alternativa (figura 3.5(d)), que parece mais adequada ao ambiente proposto, faz uso de uma organização em dois níveis. O primeiro deles é global e leva em consideração os objetos de todos os mapas armazenados no banco de dados. Esse nível é responsável pela divisão e distribuição dos fragmentos de mapas pelo sistema. O segundo nível, local a cada nó do sistema, armazena separadamente os elementos pertencentes a cada um dos fragmentos de mapas.

O diferencial dessa política se dá por manter a estrutura de distribuição independente da estrutura de armazenamento. Desta forma, a política de distribuição é única para todo o conjunto de mapas, garantindo um agrupamento espacial dos dados. Ou seja, elementos geométricos numa mesma posição no espaço, mas representados em mapas diferentes devem ser armazenados em um mesmo nó do sistema. Por outro lado, a estrutura de armazenamento de cada um dos fragmentos de mapa são independentes entre si, possibilitando um acesso eficiente a elementos de um único mapa.

Essa política será desenvolvida em mais detalhes no próximo capítulo já aplicada aos métodos de armazenamento propostos pela dissertação.

Capítulo 4

Armazenamento Distribuído de Mapas Vetoriais

O objetivo deste capítulo é introduzir métodos de armazenamento distribuído para mapas vetoriais visando a paralelização de operações sobre eles. As políticas de armazenamento propostas têm como origem métodos de acesso espaciais conhecidos na literatura. Os métodos de armazenamento distribuído propostos, de fato, constituem a principal contribuição desta dissertação.

Como já foi colocado, uma das formas de obter paralelismo na execução de operações num banco de dados, o chamado paralelismo particionado [DG92], consiste em particionar os dados e distribuí-los em vários processadores. Cada um deles torna-se, então, responsável pela aplicação de operações sobre os dados armazenados localmente, obtendo respostas parciais que numa etapa posterior são compostas no resultado final da operação.

Dentro deste contexto, este texto discute famílias de métodos de armazenamento distribuído para mapas, criados através de variações e combinações de métodos conhecidos, que oferecem uma base para a implementação distribuída de um sub-sistema de armazenamento para mapas vetoriais. Num primeiro momento, é definido um método baseado na definição de uma grade que particiona o espaço em fragmentos disjuntos. Em seguida, a estrutura é generalizada a partir de técnicas alternativas para a distribuição dos dados. Em todos os casos, o critério de distribuição se baseia na posição geográfica dos objetos. Isto significa que feições da superfície terrestre que estão geograficamente próximas tendem a ter suas representações armazenadas no mesmo processador. Ou seja, proximidade geográfica induzirá proximidade física de armazenamento, o que representa uma heurística adequada para minimizar o custo das operações geográficas.

Os métodos de armazenamento sugeridos adotam uma estratégia em dois níveis, seguindo a política sugerida na seção 3.3. No primeiro, um determinado método é usado para definir um critério de distribuição dos mapas entre os processadores. No segundo nível, um outro método é adotado para armazenar os fragmentos localmente em cada processador. A figura 4.1 mostra um esquema genérico das estruturas de armazenamento propostas.

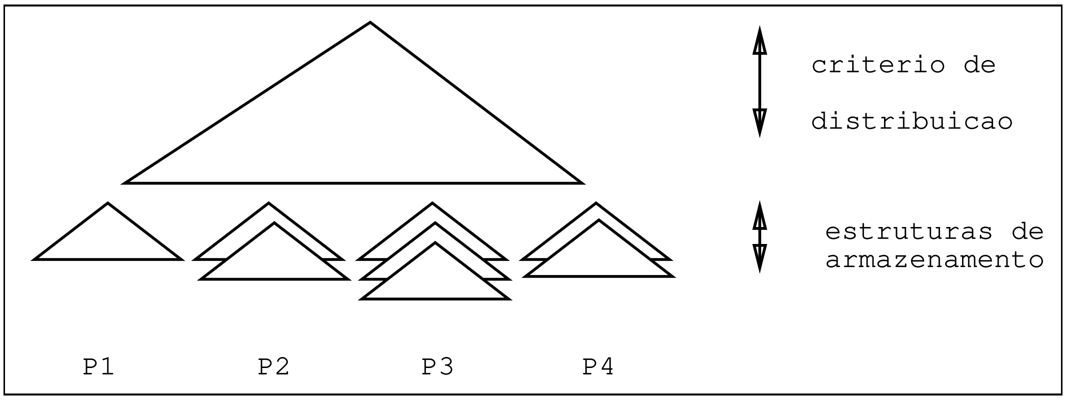

Figura 4.1: Estrutura genérica para armazenamento distribuído. Os diferentes métodos propostos implementam estruturas de dados distintas para o critério de distribuição e para o armazenamento dos fragmentos de mapas.

O critério de distribuição é responsável pela localização e gerência dos dados fragmentados. A estrutura que o define pode estar replicada em todos os processadores ou armazenada em apenas um ou alguns deles, dependendo da arquitetura global do sistema. Os métodos de armazenamento locais são aqueles propostos para o acesso a dados em ambientes centralizados, o que é possível, pois fragmentos de mapas nada mais são do que mapas restritos a um determinado limite espacial.

O método proposto é definido, então, em uma arquitetura de memória distribuída, onde cada nó do sistema é responsável por armazenar fragmentos de mapa e executar operações locais sobre estes. Considera-se que o sistema possui dois tipos de nós: de controle e de dados[1]. O nós de dados funcionam como servidores de consulta, armazenando fragmentos e executando operações locais sempre que requeridas por um nó de controle. Os nós de controle armazenam a estrutura de distribuição e fazem a in-

[1]Um mesmo processador pode desempenhar os dois papéis.

terface do método de distribuição e armazenamento com as camadas centralizadas da arquitetura. Rotinas de edição e consultas são executadas primeiramente num nó de controle que têm como responsabilidade mapear a operação geral em operações locais enviadas para os nós de dados que armazenam os fragmentos de mapa envolvidos.

A seção 4.2 descreve de forma breve um dos métodos propostos e apresenta os procedimentos básicos de (1) *recuperação* de fragmentos; (2) *inserção* de um novo mapa no banco de dados; (3) *criação* de um mapa a partir de fragmentos já alocados nos processadores; e (4) *remoção* de um mapa da estrutura. Considera-se que alterações em uma mapa armazenado, no caso geral, são feitas pela seqüência *seleção-remoção-inserção*. Este procedimento não é uma limitação, já que se pressupõe um ambiente de banco de dados estático e um módulo de edição anterior a inserção do mapa no banco de dados.

A seção 4.3 generaliza o método propondo variações do critério de distribuição baseadas em *hB-trees* e *R-trees*. A primeira ainda define uma partição do espaço representado em fragmentos disjuntos. A segunda variação é baseada na divisão uniforme dos dados e usa as heurísticas de agrupamento de dados definidas para árvores R e suas variações. A seção 4.4 faz uma série de considerações comparando as diversas alternativas.

Antes disso, porém, a próxima seção discute como mapas, seus fragmentos e os elementos que os constituem são registrados no banco de dados. A descrição do *Catálogo de Mapas* é feita de forma geral, sem depender da estrutura de dados usada.

4.1 Geração de Identificadores para Mapas, Fragmentos e Elementos de Mapas

A nível conceitual, supõe-se que o banco de dados possui uma classe de objetos chamada MAPOTECA, contendo todos os mapas.

A nível físico, essa classe é implementada por uma tabela, o catálogo de mapas, contendo para cada mapa, um par (idM, box), onde idM é o identificador único do mapa e box é seu retângulo envolvente. O mapa em si é armazenado em diversas estruturas de dados.

Esta seção endereça o problema de geração de identificadores para os diversos componentes de um mapa e, no caso de armazenamento distribuído, como gerar identificadores para os fragmentos de mapas (note que o mesmo processador pode conter mais de um fragmento do mesmo mapa).

Os mapas armazenados na MAPOTECA são registrados num catálogo global. Uma entrada no catálogo associa a cada mapa um identificador único e guarda seu retângulo envolvente. Através desta informação e da estrutura de distribuição do método, é possível, quando necessário, computar os nós que compreendem a região de cobertura do mapa, identificando onde seus fragmentos estão armazenados.

Localmente, os fragmentos de um mapa são identificados por uma tupla que inclui o identificador global do mapa (idM), um identificador único da célula de fragmentação[2] que induz o fragmento (idC) e seu retângulo envolvente (box). Ou seja, o catálogo local de cada nó do sistema associa o identificador de um mapa M ao conjunto de fragmentos de M armazenados localmente e identificados pelo identificador da célula. A estrutura de dados que implementa a noção de conjunto de fragmentos guarda uma lista de células e retângulos envolventes dos fragmentos do mapa, cada um apontando para uma estrutura de armazenamento independente.

A figura 4.2 exemplifica a estrutura dos catálogos de mapas. O processador P armazena dois fragmentos do mapa M_1 induzidos, respectivamente, pelas células c_{11} e c_{12}, apenas um fragmento de M_2, induzido pela célula c_{11} e não armazena fragmentos de M_3 (que não consta do catálogo local).

CATALOGO GLOBAL

id	Ret. Envolvente
M1	bb(M1)
M2	bb(M2)
M3	bb(M3)

CATALOGO LOCAL EM P

id	conjunto
M1	c11, bb(M11) c12, bb(M12)
M2	c11, bb(M21)

Figura 4.2: Exemplo de cátalogos global e local a um processador P.

O processo de transferência de dados entre processadores atinge a administração dos catálogos de dados da seguinte forma. A unidade a ser transferida é fragmento de

[2]Relembrando, os conceitos de fragmentação de mapas estão definidos na seção 3.1.2.

mapa, ou seja, num rearranjo da estrutura de dados, um fragmento de mapa ou um conjunto deles é transferido de um nó do sistema para outro. Quando isso ocorre, o catálogo global permanece inalterado, pois este não conhece a organização dos dados a nível de fragmento. Por outro lado, os catálogos locais aos processadores de origem e destino devem ser atualizados.

Num primeiro passo, toda a estrutura de dados que armazena o fragmento a ser realocado é transferido para o nó destino. Note que, se o método de acesso usado faz uso apenas de endereçamento relativo, essa transferência é facilitada, pois não necessita de rearranjo de toda a sua organização. A entrada no catálogo local ao nó origem referente a esse fragmento é removida, enquanto uma nova entrada no catálogo local do nó destino é criado contendo o identificador da célula da grade que induz o fragmento e seu retângulo envolvente (entrada idêntica ao armazenado no catálogo local anterior). Se já existe um fragmento do mesmo mapa armazenado neste nó, a entrada no catálogo é inserida na lista de fragmentos do mapa. O mesmo raciocínio é usado na remoção de um fragmento: se este for o único fragmento do mapa armazenado no nó do sistema, a entrada relativa ao mapa é inteiramente removida do catálogo local de origem.

Um mapa é composto por elementos geométricos que podem ser recuperados a partir de identificadores próprios. A identificação de um elemento é relativa ao mapa onde está representado. Globalmente, um elemento será identificado, então, por idM.ide, onde idM identifica de forma única o mapa onde o elemento está representado e ide identifica o elemento dentro do mapa. Esta identificação define formas de acesso direto a geometria dos objetos. O modo de acesso direto depende da estrutura de armazenamento. Em geral, o identificador do objeto aponta para a sub-estrutura que o armazena. Por exemplo, aponta para a folha de uma árvore R que armazena retângulos envolventes de polígonos e se refere a geometria exata do elemento.

De forma equivalente, as partes de um elemento fragmentado são identificados por um mesmo identificador a menos de sua localização (célula da grade em que se encontra). Ou seja, idM.idE.idC nomeia a parte do elemento fragmentado idM.idE presente no nó P_i do sistema indicado pela célula idC da estrutura de distribuição. Para recuperar um objeto identificado, por exemplo, por $M_1.l_3$, deve-se buscar as partes do elemento l_3 nos diversos fragmentos de M_1 em cada um dos processadores. No exemplo da figura 4.3, para recuperar o lago representado no mapa M, todos os processadores devem buscar as partes do lago nas estruturas de dados que armazena os fragmentos locais. A região hachurada é identificada por M.lago.c_{00}, onde M nomeia o mapa, lago identifica o elemento dentro do mapa e c_{00} a célula que induz o fragmento.

Chama-se atenção para o fato de que fragmentos de objetos só são conhecidos ao nível da arquitetura onde é inserida a função de corte e distribuição. Ou seja, sua identificação é usada apenas internamente na gerência da estrutura distribuída. Para efeito de uso pela aplicação, apenas mapas (inteiros) ou elementos de representação de geo-objetos (também não fragmentados) são efetivamente conhecidos.

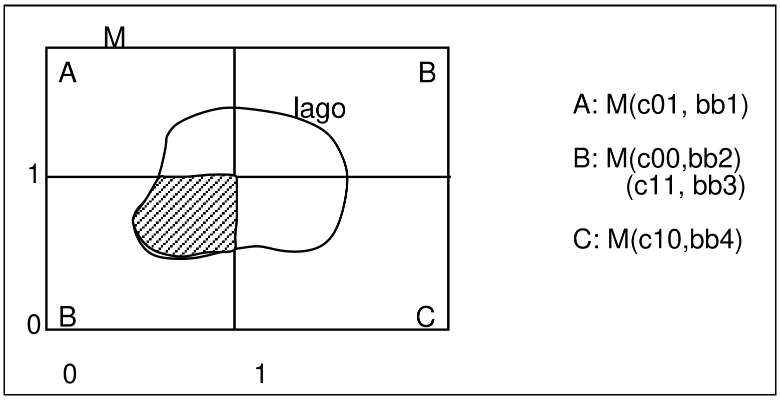

Figura 4.3: Mapa com um único elemento fragmentado e distribuído pelos nós do sistema.

Operações entre mapas são executadas a partir da composição de operações sobre fragmentos. Assim, é possível gerar um novo mapa a partir de operações locais sobre fragmentos. O mapa e seus elementos componentes são, a princípio, identificados pela composição dos identificadores dos fragmentos originais. Uma etapa posterior pode customizar esses identificadores para evitar que fiquem muito extensos.

O mapa resultante da aplicação de uma operação θ sobre dois mapas M, N é identificado por $M\theta N$. Seus fragmentos são nomeados $M.c_i\theta N.c_i$ ou $(M\theta N).c_i$ A seguir, um exemplo ilustra a geração de identificadores dos elementos do novo mapa.

Digamos que seja gerado um mapa a partir da interseção entre os dois mapas da figura 4.4. Os operadores locais aplicados nos nós A e B do sistema (células a e b) geram em A, a região mapa1.quadrado.a $\otimes$ mapa2.bola.a e em B, a região mapa1.quadrado.b $\otimes$ mapa2.bola.b. Os dois elementos têm o mesmo identificador a menos do campo que identifica sua localização (célula da estrutura de distribuição), logo são partes de um mesmo elemento fragmentado. A etapa de customização pode, por exemplo, renomear esse objeto para R.forma onde R identifica o novo mapa (registrado no catálogo de mapas na rotina de criação - seção 4.2.3.3) e forma nomeia de forma única o elemento dentro do mapa. As partes do objeto seriam R.forma.a e R.forma.b.

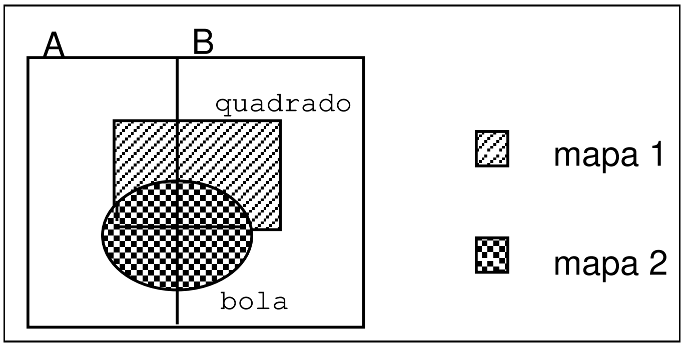

Figura 4.4: Interseção de mapas.

Da mesma forma que a estrutura que define o critério de distribuição, o catálogo global pode estar replicado em todos os processadores ou armazenados apenas em um ou alguns deles. Essa política depende da arquitetura global do sistema e deve acompanhar o critério de distribuição. De acordo com a política adotada, o sistema configura-se numa arquitetura mestre-escravo (os processadores que contém o catálogo/grade comandam o processamento de consultas) ou em uma arquitetura completamente distribuída, situação na qual qualquer um dos nós do sistema pode ser usado para executar consultas sobre o banco de dados.

4.2 Métodos baseados em Grades

Esta seção descreve brevemente uma família de métodos de armazenamento distribuído para mapas que utilizam grades para definir o critério de distribuição. A escolha do método de armazenamento local é deixada livre e deve levar em conta principalmente a otimização da execução das operações locais.

Uma grade pode ser definida de diversas formas. Uma delas baseia-se na estrutura de um *grid file* (seção 2.4.1) ou em suas extensões. Aqui a definição de uma grade G é semelhante à definição do *grid file* original [NHS84]. A definição do critério de distribuição G inclui, ainda, uma *função de alocação* $\mathcal{A}[G]$ que mapeia cada *célula* c da grade em um conjunto $\mathcal{A}[G](c)$ de processadores, no caso mais geral em que há replicação de fragmentos de mapas. Dado um mapa M, o fragmento de M induzido por c será então armazenado de forma replicada em todos os processadores no conjunto $\mathcal{A}[G](c)$. Por simplicidade, considera-se aqui que $\mathcal{A}[G](c)$ é um conjunto unitário.

A utilização de grades para representar o critério de distribuição simplifica a localização de um objeto e facilita as operações espaciais entre mapas, já que força o armazenamento nos mesmos processadores dos fragmentos de todos os mapas que sejam induzidos pela mesma célula. Ou seja, a utilização de grades agrupa, em um mesmo processador, todos os fragmentos dos diversos mapas que cobrem a mesma região do espaço.

As seções a seguir descrevem a estrutura da grade, a estratégia utilizada para melhorar o balanceamento de carga no sistema e procedimentos para a edição do banco de dados.

4.2.1 Descrição da Estrutura de Grade

Inicialmente, na definição do banco de dados, o projetista fornece, além das informações cartográficas necessárias (projeção, região de cobertura, etc...), uma grade inicial que será usada para a fragmentação dos mapas a serem armazenados. Assim, um mapa M a ser inserido na estrutura é fragmentado a partir das regiões definidas pelas células da grade.

A medida em que os diversos mapas são armazenados na estrutura, a grade pode ser refinada de modo a constituir um método de distribuição eficiente. Para isso, é implementada uma estratégia de balanceamento de carga. A seguir, descreve-se a estrutura de uma grade e seu uso como função de distribuição. Na próxima seção, discute-se uma estratégia de balanceamento responsável pelo rearranjo das células da grade pelos nós do sistema para garantir o balanceamento de carga.

A estrutura da grade G é descrita por um diretório constituído por um vetor bidimensional que define um conjunto de células c, induzindo uma função de distribuição de mapas. Um exemplo de diretório de grade está esquematizado na figura 4.5. É importante lembrar que, em um SIG, os objetos armazenados estão geo-referenciados no espaço. Com isso, a definição da grade deve acompanhar os parâmetros cartográficos definidos para os mapas que serão armazenados e implementar as devidas conversões quando necessário.

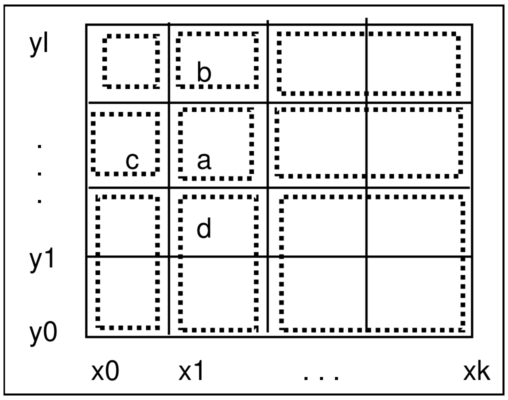

Figura 4.5: Representação planar do diretório de definição de uma grade. Os retângulos pontilhados indicam fragmentos armazenados em diferentes processadores.

Cada célula c está associada a um par (P_c, N_c), onde $P_c = \mathcal{A}[G](c)$ é o processador em que os fragmentos induzidos por c são armazenados e N_c indica a ocupação da célula (em bytes, por exemplo). O método proposto mantém a ocupação de células como referência para facilitar a gerência da estrutura e a alocação e rearranjo de células. A ocupação de cada processador é dada pela soma das ocupações das células que armazena. A expressão $G[x_i, y_i]$ denota a célula c definida pelo retângulo definido pelos pontos $(x_i, y_i), (x_i, y_{i+1}), (x_{i+1}, y_{i+1})$ e (x_{i+1}, y_i). Diversas células podem compartilhar o mesmo processador, como mostram as regiões pontilhadas da figura 4.5.

Visando melhorar o balanceamento de carga do sistema, são oferecidas, também, rotinas para o rearranjo da tabela de alocação e que podem ser usada pelo administrador do sistema explicitamente, quando necessário, e/ou pelas aplicações de tempos em tempos. Mais detalhes sobre o funcionamento destas rotinas estão discutidas na seção 4.2.2.

Diferente do objetivo do *grid file* original que armazena um número ilimitado[3] n de *buckets* de tamanho fixo, a grade aqui apresentada pretende armazenar em um número fixo de processadores uma quantidade ilimitada de dados.

A carga, porém, deve ser mantida distribuída uniformemente entre os processadores. Para facilitar a estratégia de alocação da estrutura, a grade pode ser refinada

[3]O termo ilimitado é usado para indicar que um número muito grande pode ser armazenado. Na prática, a configuração do sistema (tamanho dos discos, por exemplo) limita também esses valores.

sempre que necessário para melhor distribuir a carga e ser recomposta para evitar uma fragmentação muito grande após a remoção de parte dos dados. Para isso, são definidas políticas de refinamento e composição com comportamento semelhante às políticas usadas pelo *grid file*.

A política de refinamento sugerida divide a grade por uma linha ao longo de um dos eixos, passando pela célula a ser dividida e induzindo a divisão de todas as células ao longo de tal direção. A divisão é feita ao longo de eixos alternados sem privilegiar qualquer das direções. O critério mais simples para a localização do nível de corte seria o meio da célula, independente da distribuição efetiva de dados nela. Essa escolha, porém, não garante uma boa distribuição da carga. Deste modo, sugere-se o uso de uma heurística que vise a divisão equilibrada da célula, por exemplo, na qual a divisão é feita no ponto médio entre dois pontos extremos contidos na célula ou ainda em um ponto que origine duas células com ocupação semelhante.

Da mesma forma que existe uma política de refinamento da grade a medida que mapas são inseridos, deve ser definida, também, uma política de composição usada para juntar células quando mapas são removidos e a ocupação das células diminui. Segue-se, para isso, o critério sugerido em [NHS84] que compõe células vizinhas que quando reunidas, não chegam a 70% da ocupação máxima permitida em uma célula. A escolha das células a serem reunidas segue o critério de *buddy*[NHS84] com relação aos processadores definido num *grid file* que visa reconstituir diretórios de definição de grade anteriores. Por este método, por exemplo, a célula `a` da figura 4.5 pode ser reunida a `b` ou a `c`, mas nunca a `d`.

Outro aspecto a ser enfatizado diz respeito a elementos de um mapa que ultrapassam fronteiras de uma célula. Duas políticas poderiam ser adotadas. A primeira delas armazena o elemento que ultrapassa os limites da célula em todos os fragmentos que o intercepta. Essa forma é utilizada, por exemplo, em estruturas derivadas de PMR-quadtrees [Sam90], mas tem como desvantagem a duplicação de informações em diversos nós do sistema. No pior caso, como aparece na figura 4.6, o objeto seria armazenado em todos os nós do sistema dificultando sua gerência. A segunda política, adotada no método proposto, recorta tal objeto e cada um de seus fragmentos é alocado a uma célula com uma identificação de que é um elemento fragmentado. A linha de corte dá origem a pontos artificiais (`A` e `B` na figura 4.7) que são replicados em ambos os fragmentos. Reconhece-se um elemento fragmentado a partir da existência de pontos artificiais em sua estrutura. A identificação de elementos fragmentados será útil na resolução de operações espaciais tais como descritas no capítulo 5.

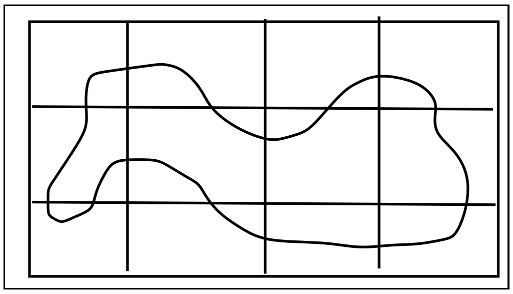

Figura 4.6: Linha extensa replicada por todos os nós do banco de dados

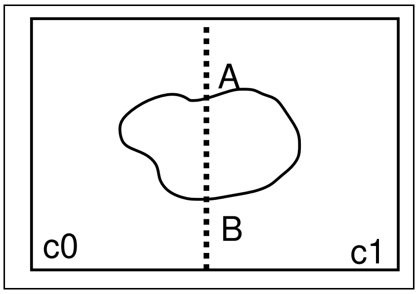

Figura 4.7: Pontos artificiais em um objeto fragmentado pela fronteira entre as células c_0 e c_1

4.2.2 Estratégia de Balanceamento

Esta seção descreve o comportamento das estratégias de alocação e balanceamento de carga usadas pelo método de armazenamento descrito acima.

A função de alocação $\mathcal{A}[G]$ associa células da grade G a processadores onde os fragmentos induzidos pelas células serão armazenados. Cada nova célula criada é associada a um processador onde os fragmentos de mapa induzidos pela célula serão armazenados. A ocupação de um processador P é dada pela soma das ocupações de todas as células armazenadas nele:

$$N_p = \sum N_c$$

para toda célula c tal que $\mathcal{A}[G](c) = P$. Uma nova célula é sempre alocada ao processador com menor ocupação do sistema, visando evitar um desbalanceamento de carga muito grande.

O critério de distribuição descrito até aqui, porém, não garante o balanceamento de carga do sistema. Para garanti-lo são disponibilizadas algumas operações elementares a serem usadas na implementação de uma política de manutenção do balanceamento de carga. As primitivas oferecidas são:

1. *transferência de células*, responsável por mover uma célula c de um processador para outro de forma a equilibrar a carga do sistema. Se não for explicitado o processador destino, a rotina move a célula para o processador menos carregado.
2. *refinamento da grade*, responsável por induzir mais um nível de fragmentação da estrutura.
3. *desfragmentação da grade*, rotina inversa à anterior, responsável por agregar células vizinhas da grade, diminuindo a fragmentação da estrutura.

As três rotinas estão descritas com mais detalhes, respectivamente, nas figuras 4.8, 4.9 e 4.10. O modelo de programação paralela utilizado em todas as rotinas em pseudo-código do texto está descrito no apêndice 2.1.4. Os algoritmos *Refina* e *Encolhe* são executados num nó de controle que guarda a estrutura de distribuição.

O administrador do banco de dados deve fazer uso dessas primitivas, de tempos em tempos, implementando manualmente o processo de rearranjo da distribuição do sistema.

A opção de não usar uma estratégia automática de rebalanceamento dentro dos algoritmos de edição da estrutura evita intensa transferência de dados entre processadores na etapa de construção do banco de dados, por exemplo. Por outro lado, rotinas de realocação específicas permitem que o administrador do banco de dados ou a própria aplicação escolham o momento adequado para o rearranjo da estrutura através do seu refinamento ou desfragmentação, deixando ainda transparente ao usuário a distribuição do armazenamento. Pode-se, também garantir o balanceamento de carga do sistema através da implementação de algumas heurísticas, visando a automatização do processo de rearranjo.

No que se segue, propomos uma heurística de rebalanceamento que trabalha com a definição de duas classes de processadores disponíveis para o armazenamento da base de dados: *ocupados* e *livres* e três parâmetros de controle:

Transferência (c,destino)

Parâmetros:
c = célula a ser transferida.
destino = processador para onde célula será transferida.

Algoritmo:
    send($\mathcal{A}[\mathcal{G}](c)$, transfer, c, destino)

Algoritmo local:
se not destino
    para todo i tal que M_i está armazenado em P
        send(P_{min}, move, $M_{i,c}$)
    $P_c = P_{min}$
senão
    para todo i tal que M_i está armazenado
        send(destino, move, $M_{i,c}$)
    P_c = destino

Comentários Adicionais:
O algoritmo é executado no nó de controle e envia um pedido de transferência de c para destino para o processador $\mathcal{A}[\mathcal{G}](c)$ que armazena c.

Figura 4.8: Algoritmo em pseudo-código do procedimento de transferência de células entre processadores.

Refina (G)

Parâmetros:
G = Diretório da grade de distribuição.
eixo = eixo pelo qual a estrutura será refinada na próxima atualização da grade.
Algoritmo:
se (eixo=x) então
 eixo $\leftarrow$ y
 calcula x_d tal que $x_c \leq x_d \leq x_{c+1}$
  e linha $x = x_d$ divide célula c em c_0 e c_1, onde $N_{c_0} \approx N_{c_1}$
 $P_{c_0} = \mathcal{A}[G](c_0)$
 $N_{c_0} = \sum \parallel M_{i,c_0} \parallel$ tal que M_{i,c_0} é fragmento do mapa M_i
  induzido pela região c_0
 transferência(c_0, P_{c_0})
 $N_{c_1} = N_c - N_{c_0}$
 $P_{c_1} = P_c$
 para toda célula c' tal que linha $x = x_d$ cruza c' e $c' \neq c$
  $G[x_{c'}, y_{c'}] = (P_{c'}, N_{c'} = \sum \parallel M_{i,c'_0} \parallel)$
  $G[x_d, y_{c'}] = (P_{c'}, N_{c'} = \sum \parallel M_{i,c'_1} \parallel)$
senão
 eixo $\leftarrow$ x
 calcula y_d tal que $y_c \leq y_d \leq y_{c+1}$
  e linha $y = y_d$ divide célula c em c_0 e c_1, onde $N_{c_0} \approx N_{c_1}$
 $P_{c_0} = \mathcal{A}[G](c_0)$
 $N_{c_0} = \sum \parallel M_{i,c_0} \parallel$ tal que M_{i,c_0} é fragmento do mapa M_i
  induzido pela região c_0
 transferência(c_0, P_{c_0})
 $N_{c_1} = N_c - N_{c_0}$
 $P_{c_1} = P_c$
 para toda célula c' tal que linha $x = x_d$ cruza c' e $c' \neq c$
  $G[x_{c'}, y_{c'}] = (P_{c'}, N_{c'} = \sum \parallel M_{i,c'_0} \parallel)$
  $G[x_{c'}, y_d] = (P_{c'}, N_{c'} = \sum \parallel M_{i,c'_1} \parallel)$

Figura 4.9: Algoritmo em pseudo-código do procedimento de refinamento da grade.

Encolhe (G, c, c_v)

Parâmetros:
G = Diretório da grade de distribuição.
c, c_v = células candidatas a se comporem

Algoritmo:
se (eixo = x) então
    eixo $\leftarrow y$
    se todo par c_0, c_1 vizinhas, separadas pela linha $x = x_d$ que separa c e c_v
        e $N_{c_0} + N_{c_1} \leq 0.7 * N_{max}$
        para todo c_0, c_1
            se $P_{c_0} \neq P_{c_1}$
                se P_{c_0} menos ocupado que P_{c_1}
                    transferência(c_1, P_{c_0})
                senão
                    transferência(c_0, P_{c_1})
            $N_{c_0} = N_{c_0} + N_{c_1}$
        retira x_d da definição do diretório G a
senão
    eixo $\leftarrow x$
    se todo par c_0, c_1 vizinhas, separadas pela linha $y = y_d$ que separa c e c_v
        e $N_{c_0} + N_{c_1} \leq 0.7 * N_{max}$
        para todo c_0, c_1
            se $P_{c_0} \neq P_{c_1}$
                se P_{c_0} menos ocupado que P_{c_1}
                    transferência(c_1, P_{c_0})
                senão
                    transferência(c_0, P_{c_1})
            $N_{c_0} = N_{c_0} + N_{c_1}$
        retira y_d da definição do diretório G

Figura 4.10: Algoritmo em pseudo-código do procedimento de composição da grade.

1. MIN, a *ocupação mínima* de um processador, usada com o intuito de evitar uma fragmentação exagerada.

2. MAX, a *ocupação máxima* aceitável para um nó do sistema dado que ainda existem processadores livres. Esse parâmetro evita sobrecarga dos nós no caso em que o banco de dados pode fazer uso de mais nós.

3. F, a *constante de desbalanceamento* aceitável. Define-se, ainda, f, o *fator de desbalanceamento*, definido pela diferença de ocupação entre os processadores mais e menos carregados (entre os nós ocupados):

$$f = N_{P_{max}} - N_{P_{min}},$$

   onde $N_{P_{max}}$ é a ocupação do processador mais ocupado e $N_{P_{min}}$ é a ocupação do processador em uso menos ocupado. Se $f >$F, as células da grade devem ser realocadas.

Todos esses fatores são definidos na criação do banco de dados e dependem da extensão e granularidade dos dados, assim como das características físicas do sistema.

A heurística de rebalanceamento proposta rearranja a estrutura em três etapas, tanto para o refinamento como para a desfragmentação da estrutura. Para refinamento, as etapas são: (1) introdução de novos processadores; (2) rebalanceamento sem refinamento; e (3) refinamento seguido de rebalanceamento. Para desfragmentação, as etapas do procedimento proposto são: (1) encolhimento da grade; (2) rebalanceamento e (3) liberação de processadores.

Considere inicialmente a heurística de refinamento. O primeiro passo é a utilização de processadores disponíveis. Ou seja, se existirem nós do sistema disponíveis e vazios e nenhum dos processadores não livres tem ocupação N_P menor que o limite MIN e algum deles tem ocupação maior que o limite MAX, a célula mais carregada de P_{max} (nó mais carregado) deve ser realocada, então, para o processador livre. Esta operação é repetida até obter uma situação em que $f \leq$ F, ou terminarem os processadores disponíveis ou a configuração sair dos limites MIN / MAX requeridos. Nota-se que esse procedimento (sem refinamento da grade) só faz sentido se mais de uma célula puder ser alocada aos processadores usados.

A figura 4.11 ilustra esta heurística. Na configuração (a) existem seis processadores disponíveis, a constante de balanceamento F é 4, MIN é 2 e MAX é 8. A configuração

(b) é formada quando a célula $G[1,1]$ é transferida para o processador F que estava vazio. A estrutura, nessa situação, continua desbalanceada, mas não existem mais processadores vazios.

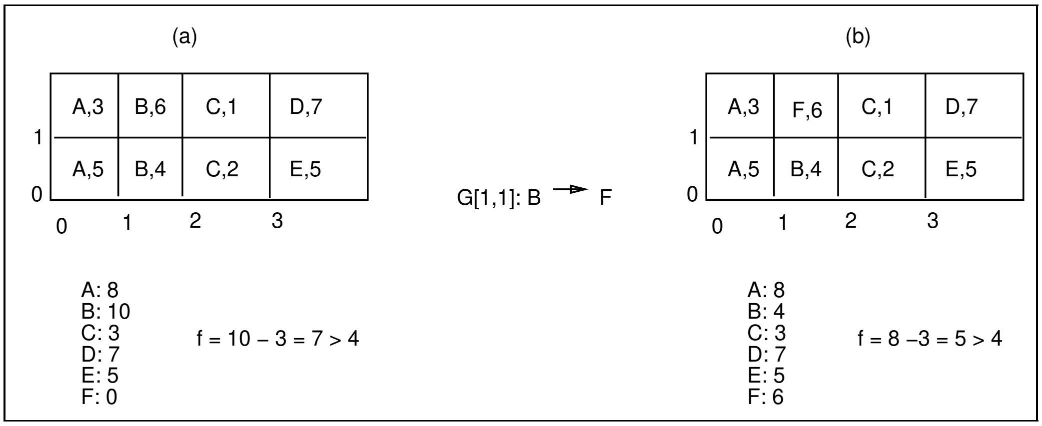

Figura 4.11: Diretório da grade G antes e depois da transferência da célula $G[1,1]$ do nó B para o nó F.

O segundo passo da heurística de balanceamento consiste em transferir células entre os processadores buscando o equilíbrio de carga pelo sistema. Observa-se que esse problema é equivalente ao de alocação de processos a processadores, objeto de estudo das áreas de algoritmo e paralelismo [HS78, CMM67, Bar93].

A figura 4.12 ilustra um procedimento simples aplicado sobre a situação final da figura 4.11. Neste exemplo, usa-se uma abordagem *gulosa* em que a célula menos ocupada do processador com maior ocupação é transferida para o processador com menor ocupação. Este procedimento é repetido até que se obtenha uma configuração balanceada. É importante que esse método seja capaz de identificar e evitar ciclos que retornem, após uma série de passos a uma configuração anterior. Uma forma de otimização possível é fazer as atualizações necessárias numa grade auxiliar e somente efetivar as transferências de dados quando se chegar a uma configuração final mais adequada que a original.

Modificando ligeiramente o exemplo, digamos que a grade encontre-se na situação descrita na figura 4.13(a). Neste caso, o processador mais carregado,D, só armazena uma única célula $G[3,1]$. Para obter uma melhor configuração de carga é necessário um refinamento da grade seguido da realocação da célula criada.

O refinamento da grade segue os critérios descritos na seção anterior. Uma das

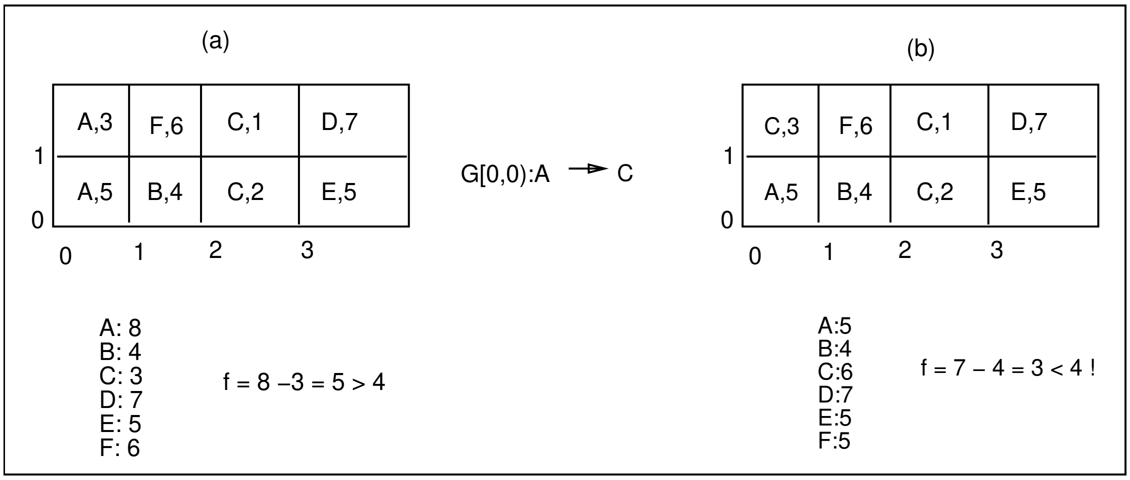

Figura 4.12: Diretório da grade G antes e depois de transferências de células entre processadores.

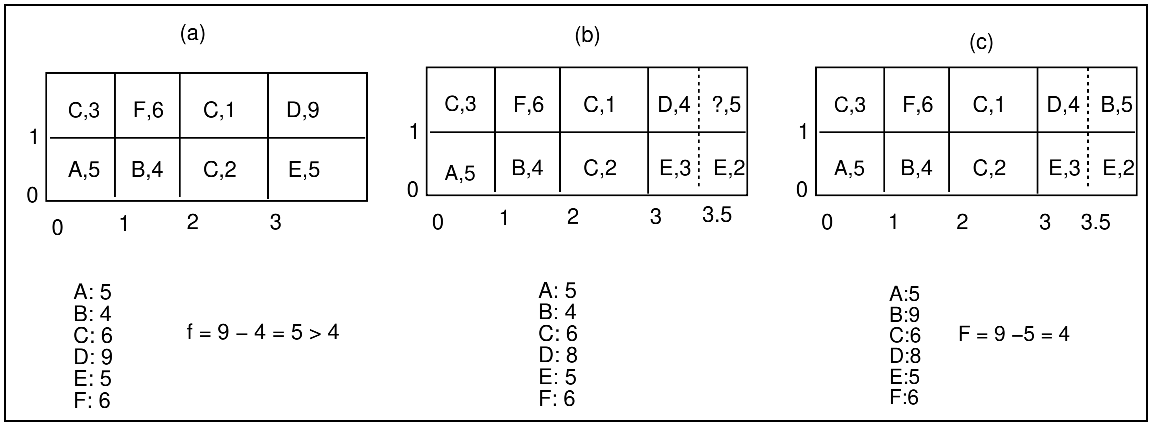

Figura 4.13: (a) Situação inicial da grade; (b) grade refinada. Em seguida suas células realocadas até alcançar (c) uma situação de equilíbrio.

células resultantes da divisão substitui a original e a outra é alocada ao nó menos carregado. A estratégia de balanceamento é então aplicada até obter uma configuração dentro do limite de desbalanceamento permitido.

Após sucessivos refinamentos da grade e uma série de remoções de mapas da estrutura, a grade pode estar excessivamente fragmentada, com poucos elementos em cada célula. Neste caso, é indicado um encolhimento da grade até que se obtenha uma configuração equilibrada, onde cada processador tenha a ocupação mínima MIN definida pelo método.

Como já foi dito, o primeiro passo para a desfragmentação da estrutura é o agrupamento de células. A heurística identifica células vizinhas à célula menos carregada de P_{min} (nó menos carregado) e que respeite o critério de composição. As células podem, ainda, ser rearranjadas de forma a obter uma melhor configuração da distribuição da grade.

Quando nenhum dos nós do sistema tem ocupação maior que MAX e existe uma configuração possível que libere pelo menos um dos processadores de forma a manter a grade dentro da faixa válida, o nó deve ser liberado. Note que a grade inicial, parte da definição do banco de dados, não deve ser desfragmentada.

A seguir, são apresentados exemplos de algoritmos em pseudo-código que implementam essa heurística, usando as rotinas de refinamento e composição já descritas.

O algoritmo da figura 4.14 resume o procedimento de rearranjo com refinamento da grade, usado após uma série de inserções de mapas na estrutura. O algoritmo 4.15 descreve a rotina de rearranjo com composição de células, usado após uma série de remoções de mapas da estrutura.

Rearranjo (G) /* com refinamento */

Parâmetros:
G = Diretório da grade de distribuição.

Algoritmo:
enquanto $(f > F)$ e $(\ \exists P = \phi)$ e (P_{max} aloca mais de uma célula) e (não $\exists N_P <$ MIN) e $(N_{P_{max}} >$ MAX)
    seja c a célula mais carregada de P_{max}
    transferência(c, P_ϕ)
enquanto $(f > F)$ e (not fim) /* ainda existe célula c em P_{max} não realocada */
    seja c a célula ainda não realocada menos carregada de P_{max}
    transferência(c, P_{min})
se $(f > F)$
    refina(G)
    rearranjo (G)

Figura 4.14: Algoritmo em pseudo-código do procedimento de rebalanceamento com refinamento da grade.

Rearranjo (G) /* com composição */

Parâmetros:
G = Diretório da grade de distribuição.

Algoritmo:
enquanto (c_v = ache_vizinha($c_{min} \in P_{min}$)
    encolhe(G, c, c_v)
enquanto $(f > F)$ e not fim /* ainda existe célula c em P_{max} não realocada */
    transferência(c, P_{min})
se todo $N_{P_i} <$ MAX
    para toda $c \in P_min$
        ache P tal que $N_P + N_c <$ MAX
        transferência(c, P)

Figura 4.15: Algoritmo em pseudo-código do procedimento de rebalanceamento com composição da grade.

4.2.3 Operações Básicas de Edição

Esta seção descreve as operações básicas disponíveis para a edição da estrutura de definição da grade de distribuição. Relembrando, as primitivas do modelo de programação paralela e os comandos básicos de armazenamento estão definidos na seção 2.1.4. Rotinas específicas usadas em um ou alguns algoritmos serão descritas sob forma de comentários adicionais à medida em que aparecerem no texto.

Os algoritmos apresentados, salvo consideração em contrário, são executados no nó do sistema onde o processo inicial é executado, um nó de controle, e sinalizam aos demais nós operações a serem executadas localmente. Considera-se que a estrutura de distribuição está implementada com o uso de um SGBD distribuído ou estrutura equivalente que seja responsável pelo controle do acesso concorrente, manutenção de integridade e da atualização distribuída da estrutura de distribuição. Essa discussão, porém, foge ao escopo do texto. Para uma configuração em que a estrutura de distribuição é centralizada ou replicada em apenas parte dos nós do sistema, a estrutura dos algoritmos continua válida, a consulta e a atualização da grade são feitas remotamente através de troca de mensagens implementadas de forma transparente ao algoritmo apresentado.

4.2.3.1 Recuperação de Fragmentos

Segundo o método descrito acima, fragmentos de mapas são armazenados em uma estrutura apropriada em processadores indicados pela grade de distribuição. O acesso aos dados é feito, então, a partir da identificação do(s) processador(es) que cobrem a área de interesse seguido da recuperação propriamente dita com o uso do método de acesso local.

A primeira operação elementar de acesso descrita retorna o fragmento de um mapa M induzido pela célula que contém o ponto (x_c, y_c).

Inicialmente, o diretório da grade G é consultado para identificar a célula c que cobre o ponto. Chega-se, então, ao par (P_c, N_c) contido na célula $G[x_i, y_i]$ do diretório, onde $x_i \leq x_c \leq x_{i+1}$ e $y_i \leq y_c \leq y_{i+1}$, sendo x_i, x_{i+1}, y_i e y_{i+1} índices do diretório que define a grade G. O fragmento requerido é um dos fragmento de M armazenados no processador P_c definido pela região dada por um retângulo de extremidade inferior esquerda (x_i, y_i) e extremidade superior direita (x_{i+1}, y_{i+1}). A figura 4.16 ilustra o exemplo.

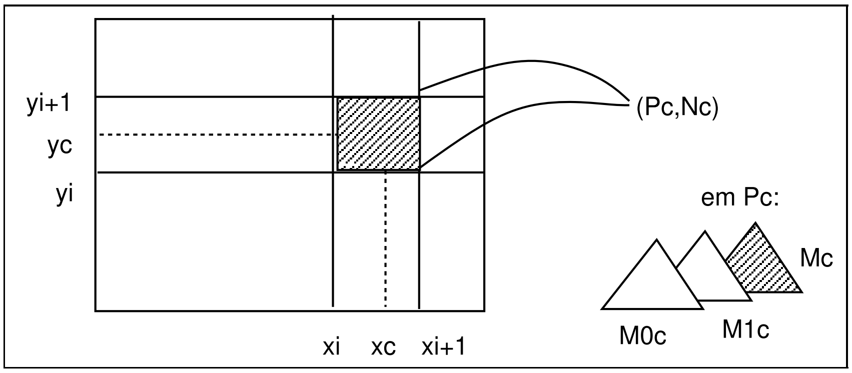

Figura 4.16: Esquema do procedimento de recuperação do conteúdo do mapa M contido na célula que contém o ponto (x_c, y_c). M_c é o fragmento de M induzido pelas fronteiras da célula c e armazenado em P_c

Outra operação elementar de acesso retorna os fragmentos de um mapa M em um retângulo definido pelos pontos (x_a, y_a) e (x_b, y_b).

Para isso, identificam-se os pontos $x_i, x_{i+1}..., x_k$, índices do diretório da grade no eixo x tais que x_i é o maior indíce de valor menor que x_a e x_k é o menor índice maior que x_b. Pelo mesmo raciocínio, identifica-se os pontos $y_j, y_{j+1}, ..., y_l$. O produto cartesiano dessas duas listas define as entradas do diretório da grade que cobrem a região de interesse. A recuperação dos fragmentos é feita, então, com o uso do método de acesso local nos processadores indicados pelas células da grade identificadas anteriormente. A figura 4.17 ilustra a situação e o algoritmo da figura 4.18 formaliza a execução da operação. Nota-se que o fragmento retornado pela rotina não contém fronteiras artificiais. Ou seja, os elementos fragmentados pelas fronteiras incluídas na área de interesse são recompostos.

4.2.3.2 Inserção de Mapas

Este procedimento insere um mapa M num banco de dados distribuído definido pela grade G.

O mapa M é dividido em um conjunto de fragmentos M_c, induzido pelo conjunto de células c da grade. Cada fragmento, então, é armazenado em uma estrutura de

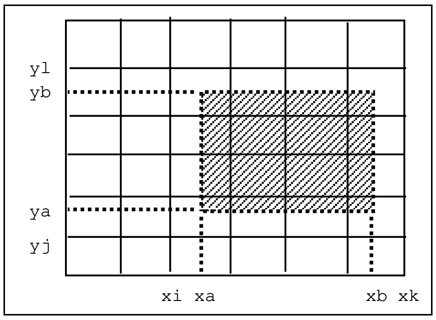

Figura 4.17: Esquema do procedimento de recuperação do fragmento do mapa M induzido pelo retângulo definido pelos pontos (x_a, y_a) e (x_b, y_b)

Acesso (M, G, p_A, p_B)

Parâmetros:
M = Mapa a ser recuperado.
G = Diretório da grade de distribuição.
p_A, p_B = Pontos de definição da área de interesse, definidos por (x_A, y_A) e (x_B, y_B), respectivamente.

Algoritmo:
ache x_i tal que $x_i \leq x_a$ e não existe x tal que $x_i < x \leq x_a$
ache x_k tal que $x_k \geq x_b$ e não existe x tal que $x_b < x \leq x_k$
ache y_j tal que $y_j \leq y_a$ e não existe y tal que $y_j < y \leq y_a$
ache y_l tal que $y_l \geq y_b$ e não existe y tal que $y_b < y \leq y_l$
para toda célula $c : G[x, y]$ tal que $x_i \leq x < x_k$ e $y_j \leq y < y_l$ /* x e y são índices do diretório G */
    send(P_c,get,(x, y), (next(x), next(y))
/* construção da resposta */
Mapa = vazio
enquanto tem mensagem
para toda célula $c : G[x, y]$ tal que $x_i \leq x < x_k$ e $y_j \leq y < y_l$ e not fim
    receive(P_c, read, aux)
    Mapa = Mapa $\cup_\oplus$ aux
retorna Mapa

Comentários Adicionais

---

next(v) $\rightarrow$ retorna o índice do diretório imediatamente acima do índice v.
A operação receive não bloqueia o *loop* e este é executado até que o processo tenha recebido
    o conteúdo de todas as células (fim = true).
O operador $\cup_\oplus$ implementa a operação de desfragmentação definida na seção 3.1.2.

Figura 4.18: Algoritmo em pseudo-código do procedimento de recuperação de um fragmento de mapa armazenado na grade.

armazenamento própria no processador P_c definido no par (P_c, N_c) da célula c. O número N_c que indica a ocupação da célula é atualizado, somando a ocupação do fragmento M_c a ocupação pré-existente.

O mapa M deve ser registrado no catálogo do banco de dados com a definição de um identificador único que identificará também os fragmentos locais.

O algoritmo na figura 4.19 resume o procedimento. A figura 4.20 ilustra um passo da inserção de um mapa M numa grade que já armazena o mapa M'. Note que a grade resultante ficou desbalanceada. A execução da rotina de rearranjo seria indicada nesse ponto, a menos que se esteja numa etapa em que uma série de mapas ainda estejam para ser inseridos.

Insert (M, G)

Parâmetros:
M = Mapa a ser inserido.
G = Diretório da grade de distribuição.

Algoritmo:
```
para cada célula c ∈ G
    gera M_c tal que M_c é fragmento de M induzido por c
    N_c = N_c+ || M_c ||
    send(P_c,store, M_c)
```

Comentários Adicionais:
$\| x \|$ = ocupação do fragmento x

Figura 4.19: Algoritmo em pseudo-código do procedimento de inserção de um mapa na grade.

4.2.3.3 Criação de Mapas a partir de Fragmentos

Um mapa pode ser inserido no banco de dados de forma *bottom-up* a partir de fragmentos criados localmente em função da execução de operações sobre mapas pré-existentes.

Cada processador possui uma área de trabalho para fragmentos temporários. No processo de criação de mapas, esses fragmentos são armazenados localmente. As células de G devem ser atualizadas com o novo número de elementos nas células que

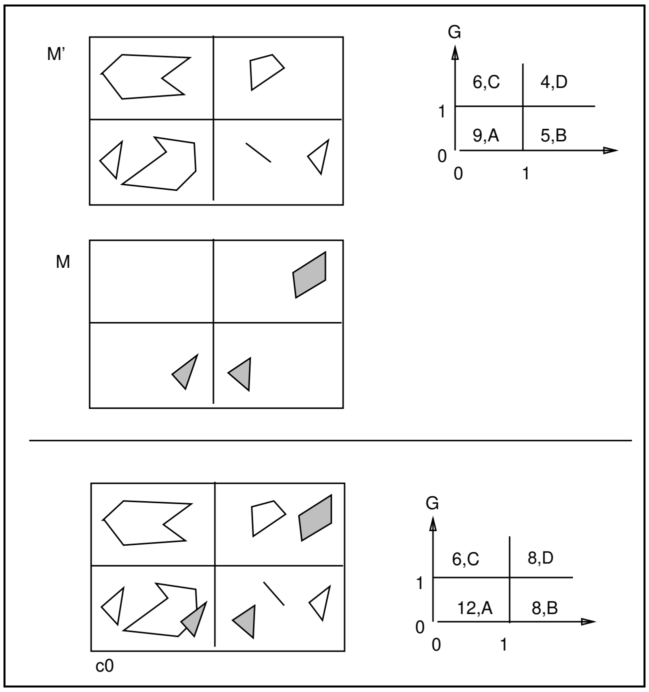

Figura 4.20: Esquema da inserção do mapa M na grade G que já armazena o mapa M'. A estrutura final indica desbalanceamento de carga no sistema

.

contêm os fragmentos criados. Novamente, é possível que essa operação resulte numa situação de desbalanceamento de carga.

O algoritmo na figura 4.21 resume o procedimento a ser seguido. Nota-se que o procedimento é bem parecido com o de inserção de mapas. A diferença é que aqui os fragmentos não são transferidos para outros processadores.

Cria (M, G)

Parâmetros:
M = Mapa a ser criado.
G = Diretório da grade de distribuição.

Algoritmo 1:
para cada processador P
    send(P, create, myself)

Algoritmo local (executado em cada P):
para cada celula c em P
    identifica M_c
    $N_c = N_c + \| M_c \|$
    armazena M_c em P

Comentários Adicionais:
armazena $\rightarrow$ rotina do método de acesso local
$\| x \| \rightarrow$ retorna o tamanho de x

Figura 4.21: Algoritmo em pseudo-código do procedimento de criação de um mapa na grade a partir de fragmentos locais. Note que a grade resultante pode ficar desbalanceada.

O mapa M gerado é registrado no catálogo da base de dados. Para isso é gerado um identificador único e o retângulo envolvente do mapa calculado pelo processador de controle a partir da união dos retângulos envolventes locais.

4.2.3.4 Remoção de Mapas

Uma mapa M é removido da estrutura com a remoção de todos os seus fragmentos dos processadores indicados pelas células que os contêm e com a atualização da grade G, se necessário.

A partir do catálogo de mapas e do retângulo envolvente de M nele definido, identifica-se os processadores que armazenam fragmentos do mapa. Localmente, cada um desses processadores identifica os fragmentos pertencentes ao mapa M e libera o espaço por ele ocupado. O par (P_c, N_c) de cada célula c da grade G é atualizado conforme a quantidade de objetos removidos. O registro referente a M no catálogo de dados é retirado.

A figura 4.22 detalha o procedimento.

Remove (M, G)

Parâmetros:
M = Identificador do mapa a ser removido.
G = Diretório da grade de distribuição.

Algoritmo:
para cada processador P_c tal que $c \cap ret(M) \neq \phi$
    send(P_c, remove, M)

Algoritmo local (a ser executado em todo P):
para cada celula c tal que $P = P_c$
    remove (M_c)
    $N_c = N_c - \| M_c \|$

Comentários Adicionais:
A rotina *ret(M)* retorna o retângulo envolvente de M.
A rotina *remove(N)* retira o fragmento N da estrutura de armazenamento local.

Figura 4.22: Algoritmo em pseudo-código do procedimento de remoção de um mapa da grade.

4.3 Métodos Alternativos

É possível definir uma série de variações para o método de armazenamento distribuído proposto. Essa seção analisa métodos alternativos a partir da variação do critério de distribuição do espaço. Por serem semelhantes aos procedimentos de edição descritos na seção anterior, apenas refletindo as variações da estrutura, será omitido o detalhamento das rotinas de edição e rearranjo deste método. O objetivo dessa seção é analisar a generalização do método proposto com o uso de diferentes critérios de distribuição.

4.3.1 Métodos baseados em Árvores hB

A primeira variação generaliza a definição de critério de distribuição de fragmentos disjuntos. A estrutura usada na seção anterior permite apenas refinamentos da grade ao longo de um dos eixos do diretório. Com isso, a divisão de uma célula induz a fragmentação de toda a estrutura, apesar de não obrigar a transferência de todas as novas células para outros nós do sistema. Esse efeito pode ser evitado, mantendo a idéia básica de partição do espaço em células de fragmentação disjuntas, com o uso de uma árvore hB como critério de distribuição.

A grade G definida na seção 4.2 é, então, substituída por uma árvore hB ligeiramente modificada em relação a definição original resumida na seção 2.4.1. Os nós de índice mantém-se como na definição original, mas os nós de dados que representam as células de fragmentação do método, não apontam diretamente para os dados armazenados. Ao invés de dados, a *kd-tree* de um nó de dado da *hB-tree* armazena o par (P_c, N_c) onde, análogamente às células da grade G, P_c indica o processador responsável por armazenar os fragmentos dos mapas definidos pela célula (bloco ou bloco furado da *hB-tree*) e N_c, a ocupação da célula. Novamente, o processador P_c é definido por uma função de alocação $\mathcal{A}[hB]$ que associa os blocos da árvore a um conjunto de processadores. A ocupação N_c é usada pelas rotinas de edição do método para refinar a estrutura sempre que necessário. O balanceamento de carga é garantido por essas rotinas através do uso de heurísticas apropriadas semelhantes às definidas em 4.2.2.

Os fragmentos de mapas induzidos pelos blocos da estrutura da árvore hB são armazenados no processador P_c por um método de acesso qualquer. Novamente a escolha da estrutura de armazenamento deve ter em mente a otimização dos procedimentos de acesso e operações espaciais locais.

A figura 4.23 exemplifica o uso de uma árvore hB como critério de distribuição do método.

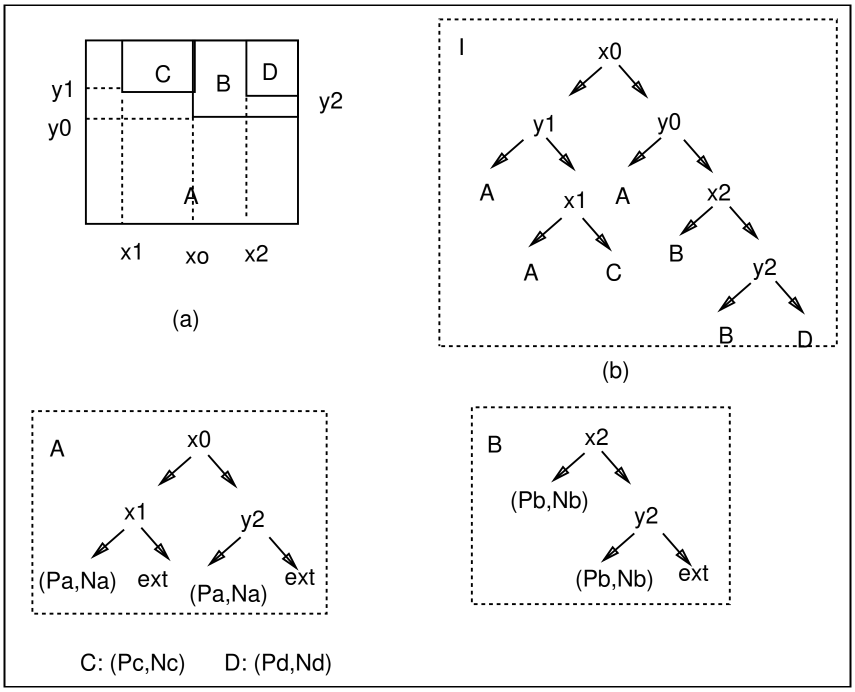

Figura 4.23: (a) Espaço particionado. (b) nó de índice da *hB-tree*. Os demais nós são os nós de dados representando as células de fragmentação da estrutura.

Esse método é semelhante ao anterior no que diz respeito à fragmentação em células disjuntas. O acesso aos dados não é tão simples, pois exige que sejam percorridas as *kd-trees* para a identificação da célula em que um dos elementos está representado.

Por outro lado, é mais natural embutir, neste caso, as heurísticas de balanceamento de carga do sistema na própria edição da estrutura, a partir de alguns parâmetros definidos, tais como o tamanho máximo de um nó (dado que existem processadores vazios), o tamanho mínimo de um nó e um fator de desbalanceamento permitido. Note que as heurísticas usadas são semelhantes às definidas na seção anterior.

Da mesma forma que no método baseado em grade, o critério aqui definido fragmenta o espaço modelado induzindo fronteiras artificiais que podem interceptar elementos geográficos representados no mapa armazenado. Novamente, estes elementos são fragmentados e cada uma de suas partes armazenada no nó apropriado.

4.3.2 Métodos baseados em Árvores R

Uma outra variação possível parte do conceito de divisão uniforme dos dados usado na definição de árvores R e suas variantes como critério de distribuição que oferece balanceamento de carga automaticamente.

Suponha que seja dada uma coleção de mapas M. O armazenamento de tal conjunto é obtido a partir da construção de uma única árvore R_M[4] contendo todos os elementos de todos os mapas. A árvore R_M é cortada num determinado nível. Sua parte superior define o critério de distribuição e as sub-árvores inferiores ao ponto de corte induzem uma floresta F.

As árvores do conjunto F são, então, distribuídas pelos nós de dados e os fragmentos de mapa nela contidos são armazenados localmente em estruturas de armazenamento próprias.

Em outras palavras, o critério de distribuição é definido por uma árvore R chamada *árvore de distribuição* e denotado por D. O número de folhas da árvore de distribuição é limitado e associado diretamente a processadores. Localmente cada fragmento é armazenado, por exemplo, em uma árvore R própria. A distribuição de carga da estrutura é definida pelo critério de quebra de nós da árvore de distribuição e implicitamente balanceado.

Na verdade D não é necessariamente uma árvore, pois o número de folhas (processadores) não deve ser muito grande para não espalhar demais os dados armazenados e perder o ganho de desempenho por excesso de comunicação entre nós. Com isso, a estrutura D pode ser simplesmente uma lista encadeada, um vetor ou qualquer outra estrutura de dados simples. Neste texto, mantém-se a árvore como forma de obter algum ganho no acesso à estrutura.

Uma árvore D difere ligeiramente da definição tradicional das árvores R como descrito a seguir. Seja L uma folha de D e (T, B), uma entrada em L, onde T aponta

[4]A árvore R_M não é construída realmente por inteiro, pois apenas os níveis até o ponto de corte serão mais tarde considerados.

para um nó filho e B é o retângulo envolvente associado a este filho. Este par é substituído pela tripla (P, B, N) tais que P é o conjunto de processadores contendo réplicas dos fragmentos associados ao nó e N é a ocupação do processador associado a folha.

Uma característica que difere essa estrutura da analisada na seção 4.2 é a possibilidade de existência de folhas da árvore de distribuição que se sobrepõem. No caso geral, em que D é uma árvore R simples ou uma R*, diferentes folhas podem cobrir áreas que se interceptam. Neste caso, fragmentos que passem por tal região podem existir em qualquer um dos nós do sistema envolvidos. Esse fato não constitui limitação no acesso aos dados (já não era no caso centralizado), pois o acesso aos diferentes processos é feito concorrentemente. Porém, traz um novo fator na execução de operações que envolvam mais de uma mapa, pois pares de objetos que satisfazem a restrição da consulta podem estar em processadores diferentes, incorrendo na necessidade de maior comunicação entre os processadores. As operações entre conjuntos podem ser facilitadas com o uso de uma política de manutenção dos nós da árvore de distribuição que induza agrupamento espacial dos objetos.

Por outro lado, nós de uma árvore R são definidos a partir do retângulo envolvente de objetos componentes dos mapas. Assim, a princípio, não existem elementos fragmentados como na grade. Apesar desse fato aparentemente simplificar os algoritmos, já que não se preocuparão com a existência de partes de um mesmo elemento em diferentes processadores, ele pode induzir particionamentos ruins ao lidar com mapas com elementos muito extensos. Por exemplo, na partição da figura 4.24 (a), operações sobre a linha do nó A (o rio Amazonas, p.ex.) exigem comunicação com os nós B e C. Este mesmo exemplo, numa grade definida como na figura 4.24 (b) demandaria apenas a execução local em dois processadores.

A recuperação de elementos de um mapa armazenado de forma distribuída a partir da árvore de distribuição D é feita com a identificação dos processadores candidatos a conterem fragmentos do mapa seguido do acesso a árvore R que armazena os fragmentos localmente.

O primeiro passo é identificar as folhas da árvore D cujo retângulo envolvente intercepta a área de interesse. Para isso é usado o procedimento padrão de acesso a árvores R que a percorre a partir da raiz apenas pelas sub-árvores que cobrem a área de interesse. A recuperação é feita, então, através do método de acesso local nos processadores indicados pelas folhas de D.

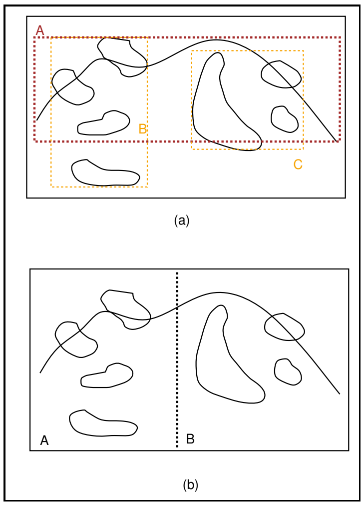

Figura 4.24: As figuras mostram particionamentos possíveis sobre um banco de dados que armazena dois mapas e os nós do sistema que armazenam cada uma das partes.

4.4 Outras Considerações

Uma análise das seções 4.2 e 4.3 indica que todos os métodos de armazenamento discutidos são muito parecidos. A diferença entre eles, ocasionada pelas políticas definidas pelos critérios de distribuição devido a existência ou não de particionamento total e superposição e suas conseqüencias pode ser observada nos seguintes termos:

As estruturas foram propostas a partir de duas visões distintas. A primeira, usada pelos métodos baseados em grade e em árvores hB, propõe o particionamento total do espaço modelado, enquanto a segunda, usada por aqueles baseados em árvore D, divide o espaço segundo a distribuição efetiva dos objetos (a densidade da distribuição dos objetos se reflete diretamente na estrutura). Assim qualquer novo fragmento inserido na estrutura de grade encontra uma célula candidata a armazená-lo (essa pode ser, no máximo, dividida e redistribuída), enquanto na estrutura baseada em árvore R, usa-se heurísticas espaciais e de balanceamento para escolher o nó onde o fragmento será alocado.

As definições da grade e da árvore hB não permitem superposição das áreas cobertas por mais de uma célula. Esse fato simplifica os procedimentos de execução da maioria dos operadores (os que não induzem alteração no retângulo envolvente considerado) sobre a estrutura. Já o método baseado em árvores balanceadas, no caso geral em que a estrutura de distribuição é dada por uma árvore R genérica que permite superposição de nós, tem que prever um tratamento especial para as áreas de interseção de áreas cobertas por mais de um ramo da árvore.

Nos métodos baseados em partições disjuntas, elementos que interceptam as fronteiras da célula são fragmentados, diferente do método que permite superposição, construído a partir dos retângulos envolventes dos elementos. O fato da estrutura baseada em árvores R não fragmentar objetos não indica que esta estrutura é mais simples do que a baseada em grades. A existência de objetos longos, mesmo que armazenados numa estrutura apropriada (V-tree [MCD94]), pode tornar a distribuição a nível de mapa menos eficiente, já que não particionaria, por exemplo, um rio muito extenso, definido por uma única linha. Algumas heurísticas na execução de operações podem minimizar esse problema, por exemplo, selecionando, agora sim, o fragmento do objeto que passa por uma determinada região e movendo-o para o nó que contém os elementos a serem com ele relacionado. Outra forma de abordar o problema é usar uma variante da árvore R que recorta objetos. Exemplo disso são as árvores R^+ que não permitem sobreposição de nós e por isso dividem objetos.

A realocação de fragmentos em qualquer um dos métodos não inviabiliza seu uso, já que um banco de dados geográfico tende a ser estático e a realocação pode ser feita concorrentemente com outras operações, adaptando-se, por exemplo, o procedimento de rearranjo proposto em [ELS95]. Além disto, por assumirmos que os processadores formam um *cluster*, a transferência de dados entre eles é rápida.

Novamente, em todos os métodos, o número de nós do sistema utilizado disponibilizados para armazenamento não deverá ser muito grande, mesmo quando possível. O dimensionamento do ambiente de armazenamento deve levar em consideração a natureza dos dados. Ou seja, o número de nós usados, fornecido na definição do banco de dados, deve ser função do tamanho do espaço modelado e da granularidade dos dados. Essa limitação decorre do fato de que uma fragmentação muito grande (poucos objetos em cada nó) exigiria um maior grau de comunicação entre os processadores e ainda induziria a um índice de fragmentação de objetos grande.

Em outras palavras, a partir das métricas definidas na seção 2.1.2, infere-se que a arquitetura proposta apresenta um bom *scale-up*, pois o aumento de nós disponíveis deve aumentar a quantidade de dados processados em um mesmo período. O *speed-up*, porém, é limitado, pois aumentar o número de nós do sistema não diminui necessariamente o tempo de processamento de uma mesma massa de dados.

Finalmente, a última observação a ser feita é sobre a necessidade de aplicar um sistema de controle de concorrência na gerência das estruturas de distribuição. Até aqui foi considerada a existência de um sistema gerenciador de banco de dados ou uma biblioteca para gerência de estruturas de dados em ambientes multi-processados. Essa biblioteca torna transparente a política de controle de integridade que garante o acesso de diversos processos a uma estrutura de distribuição global. Note que se a arquitetura escolhida exigir, a estrutura de dados deverá estar replicada em vários nós do sistema evitando acesso remoto a mesma, mas embutindo mais uma preocupação para manter as estruturas atualizadas e coerentes entre si.

Capítulo 5

Processamento de Operações Espaciais

Dentro do objetivo de melhorar o desempenho na manipulação de objetos em um SIG e do contexto da arquitetura vetorial apresentada em 3.3, este capítulo preocupa-se com a resolução de operações geográficas pertinentes às representações vetoriais de objetos. Em outras palavras, ela trata da resolução das cláusulas espaciais das consultas, num ambiente em que a recuperação de objetos por atributos convencionais é tratada por um sub-sistema gerenciador de banco de dados convencional.

Não se pretende apresentar algoritmos que cubram todas as classes de operações existentes, ou formular um procedimento geral para a geração de um interpretador ou compilador de consultas. A idéia é propor e analisar soluções para algumas operações representativas que servem de base para demonstrar o uso das estruturas propostas no capítulo anterior. A partir da síntese de uma arquitetura básica para a resolução de operadores, será possível desenvolver algoritmos que resolvam toda uma gama de operadores semelhantes.

Para facilitar a compreensão da arquitetura, o texto está permeado de exemplos. Normalmente, os exemplos são constituídos por consultas formadas em linguagem natural seguida por uma pequena explicação de como se dá sua resolução. Note que, em geral, vários planos de execução distintos resolvem o problema. O texto apresenta uma solução que faça uso do conceito que se deseja apresentar, não necessariamente a forma que seria escolhida pelo otimizador do SGBD para todos os casos.

Em outras palavras, o objetivo deste capítulo é ilustrar o uso adequado do método de armazenamento distribuído proposto como suporte na resolução de consultas geográficas. A idéia é discutir alguns conceitos relevantes através de um conjunto expressivo, porém não extensivo, de operações exemplo.

A seção 5.1 define, de modo geral, as operações espaciais abordadas no texto, enquanto que as seções 5.2 a 5.4 tratam da síntese dos algoritmos sobre as estruturas de armazenamento distribuído propostas. Os procedimentos delineados são gerais, pressupondo um critério de distribuição genérico que fragmenta elementos e permite superposição do espaço definido pelas células de fragmentação.

5.1 Classes de Operações Espaciais

Responder a questões complexas, a partir de dados armazenados, é um dos papéis de um SIG. É necessário, então, que uma série de operadores sejam disponibilizados para que a partir de um conjunto de objetos, obtenha-se outros tipos de informação sem que essas sejam explicitamente armazenadas na base de dados. Ou seja, um SIG deve oferecer um conjunto de operações que manipulam de diversas formas os diferentes tipos de objetos que armazena. Dentro deste conjunto, identificam-se operações específicas para consultas às bases de dados a partir de restrições aplicadas às representações dos objetos, ou seja, aos seus atributos espaciais.

Em geral, o processamento sobre a geometria dos objetos não requer o conhecimento de sua semântica. Em outras palavras, o algoritmo executado para identificar ruas transversais a uma avenida é o mesmo que o usado para descobrir os afluentes de um rio principal. Deste modo, é possível identificar operadores básicos que, executados sobre a geometria dos objetos, resolvem as cláusulas espaciais das consultas formuladas pela aplicação.

A referência [Gut94] classifica a álgebra de manipulação espacial em 4 grupos: seleção espacial, junção espacial, aplicação de funções espaciais (mapeamento de temas, filtros...) e outros (que incluem *overlay*, fusão...). Já [BKS93] divide as classes de consultas espaciais segundo o número de argumentos da operação, ou seja, identifica dois casos possíveis: operadores que geram resultados a partir de apenas um mapa (*single scan queries*) ou de mais de um (*multiple scan queries*). Esse critério de classificação das operações distingue, na verdade, operações que usam apenas uma ou mais de uma estrutura de dados para obter o resultado pretendido. É possível, ainda, refinar essa classificação distinguindo operações que geram como resultado um novo mapa das que retornam valores numéricos ou tuplas formadas por elementos primitivos (elementos geométricos ou valores numéricos).

O primeiro tipo de operação analisada, denominada *seleção espacial*, recupera objetos que satisfazem um dado predicado P espacial. Ela pode retornar um sub-conjunto do mapa em que é aplicada ou um valor calculado a partir dele, por exemplo, a área de um determinado tema ou os elementos contidos numa região geométrica do espaço.

Em outras palavras, uma seleção espacial pode gerar resultados de dois tipos: (1) um mapa formado por conjuntos de objetos, fragmento do mapa original que obedece a um predicado espacial, ou (2) valores numéricos calculados a partir de considerações sobre um mapa. Alguns exemplos do primeiro tipo: (a) "*retorne as cidades no Estado do Rio de Janeiro.*" - essa cláusula pode ser traduzida pela seleção de pontos (que representam as cidades em um dado mapa) interiores à região que representa o Rio de Janeiro; (b)"*retorne todos os rios que interceptam uma janela de consulta.*" - essa cláusula seleciona as linhas que representam rios e que passam pela janela de interesse; (c)"*encontre as cidades a x distância de São Paulo.*" - se as cidades (incluindo São Paulo) estão representadas por pontos, a operação retorna os pontos interiores ao círculo de raio x a partir de São Paulo. Um exemplo do segundo tipo é o cálculo da área correspondente a um tema num mapa temático ou do perímetro da fronteira de um país, onde normalmente é selecionado um sub-conjunto do mapa original seguido da inferência de algum valor a partir deles.

A mais simples dessa classe de consultas é a *window query*, na qual é retornada uma visão de um mapa restrito a uma janela pré-definida. Em outras palavras, essa operação retorna todos os objetos representados no mapa que estão no interior de uma janela de consulta (normalmente definida pelas extremidades de um retângulo). Uma série de métodos de acesso espaciais [BKSS90, Sam90, MCD94] foram propostos com o objetivo de armazenar índices espaciais que otimizam a resolução dessas operações. O acesso às geometrias armazenadas se faz através da identificação de sub-estruturas que cobrem a área definida na janela de consultas de forma recursiva até que seja feita a recuperação dos dados desejados. O procedimento de execução deste tipo de operação sobre os métodos de armazenamento do capítulo 4 está descrito em 5.3.

O segundo tipo de operação a ser analisado, denominado *junção espacial*, relaciona objetos de dois mapas a partir de relacionamentos topológicos ou geométricos entre os mesmos.

Uma outra forma de enxergar esse problema, pode ser a definição de outra operação muito freqüente que define a superposição de mapas (*map overlay*) gerando um novo mapa a partir da superposição de dois outros. Apesar dessa operação ser encontrada de forma independente na literatura [Wan93], ela nada mais é do que a generalização de uma junção espacial que gera um novo mapa sem que seja feito algum tipo de restrição. Normalmente, autores oriundos da área de geometria computacional identificam essa operação por si só, enquanto trabalhos na área de banco de dados tendem a vê-la como uma junção espacial que seleciona objetos a partir da superposição entre dois mapas.

Novamente a resposta dessa operação pode ser um novo mapa ou ainda um conjunto de pares constituídos por elementos dos dois mapas originais e/ou inferidos a partir deles. Essas operações dão suporte a consultas do tipo: *"selecione todos os pares de estradas federais e rios que se cruzam".* Essa seleção é definida por uma operação de junção que retorna elementos do primeiro mapa (estradas) que se relacionam por um operador de cruzamento com elementos do segundo mapa (rios). Outros exemplos são: (a) *"indique as cidades e os países aos quais pertencem"*; (b) *"para cada rio, encontre cidades a menos de x distância."*; (c) *"gere o mapa de potencial agrícola a partir da interseção dos mapas de solo e clima (map overlay)."*

Exemplos de predicados espaciais frequentes são a interseção, o cruzamento, a inclusão e a proximidade. Junções podem ser definidas sobre a geometria exata dos objetos ou sobre aproximações sobre eles (por exemplo, sobre o mínimo retângulo envolvente). Esse último pode ser, inclusive, usado como filtro preliminar na execução de junções do primeiro tipo [BKS93].

Considerando que cada um dos mapas tem sua própria estrutura de armazenamento e recuperação, a dificuldade embutida nesse tipo de operação está em encontrar procedimentos que relacionem ambas as estruturas de forma eficiente. O desempenho desse tipo de operação costuma ser mais crítico do que as consultas de seleção descritas anteriormente, já que numa junção espacial pode ser necessário acessar os objetos armazenados diversas vezes e ainda testar o relacionamento entre eles, resultando em operações geométricas complexas. Ou seja, as junções espaciais comuns num sistema de geoprocessamento têm seus gargalos de desempenho na recuperação de objetos (lembre que um objeto longo pode não caber por completo na memória principal resultando em n acessos a disco) e no teste do predicado de relacionamento espacial sobre geometrias complexas. Além disso, técnicas mais eficientes (*hash, sort-merge*) para a execução de junções relacionais não se adequam a dados espaciais sem que haja pelo menos um custo de mapeamento dos elementos bi-dimensionais em uma ordenação uni-dimensional.

A próxima seção descreve a síntese de algoritmos para a resolução das operações especificadas acima pressupondo mapas armazenados segundo um dos métodos descritos no capítulo anterior.

5.2 Esquema Geral para Processamento Distribuído de Operações Espaciais

Em primeiro lugar, propõe-se uma arquitetura genérica para a resolução de operações sobre as estruturas propostas pelos métodos de armazenamento distribuído introduzidos no capítulo anterior. Nas seções subseqüentes, são propostos métodos para a resolução de operações espaciais, respectivamente, com um ou mais argumentos a partir da estrutura proposta.

Em termos gerais, o método de armazenamento de mapas introduzido no capítulo 4 está estruturado em dois níveis, um que define a distribuição de fragmentos de mapas pelo sistema e o segundo, responsável pelo armazenamento local dos objetos.

Da mesma forma, o procedimento para a resolução de operações sobre essa estrutura deve, então, trabalhar em dois níveis, um responsável pelo mapeamento da operação requerida em operações sobre os fragmentos distribuídos e outra responsável pela execução local desses operadores. Na verdade, a arquitetura proposta pode ser resumida em três etapas, respectivamente, *distribuição*, *execução* e *composição*, representadas pelos retângulos da figura 5.1.

O plano de execução de cada uma delas pode variar, permitindo configurações diversas de acordo com a complexidade da operação. Considere $P_Q = P_1; P_2; ...P_n$ um plano de execução seqüencial que resolve uma consulta Q a partir dos procedimentos $P_1, P_2...P_n$ executados seqüencialmente. Do mesmo modo, considere $P_Q = P_1//P_2//...P_n$ um plano de execução que resolve Q a partir da execução dos processos $P_1, P_2...P_n$ em paralelo.

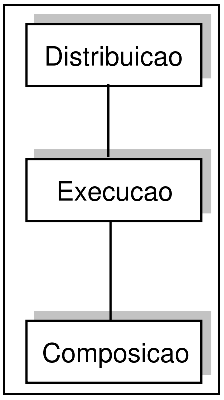

Figura 5.1: Arquitetura para a resolução de operações distribuídas.

Usando esta notação, o procedimento mais simples para a resolução de operações sobre os métodos propostos é dado por

$$P_Q = P_1; (P_{21}//P_{22}//...P_{2n}); P_3$$

A execução de P_1 corresponde a etapa de distribuição, que é responsável pelo mapeamento da operação sobre mapas em operações sobre os fragmentos dos mapas. A partir da estrutura de distribuição específica do método de armazenamento usado são identificados os nós do sistema responsáveis pelo armazenamento dos fragmentos envolvidos na consulta.

Em seguida, na etapa de execução, cada processador i identificado executa o operador local dado por P_{2i} sobre os fragmentos que armazena e envia os resultados para o nó de controle. O operador local pode ser idêntico ao operador centralizado equivalente, ou exigir processamento adicional, conforme será visto mais adiante nesta seção.

Finalmente, a última etapa de composição, definida por P_3, compõe as diversas respostas parciais, gerando o resultado final da consulta. Novamente, o nível de complexidade dessa etapa depende da semântica da operação. Por exemplo, no cálculo da área coberta por um tema, a composição da resposta consta de uma simples adição dos resultados parciais, independente da existência de objetos fragmentados. Essa etapa é, ainda, responsável pela resolução da consulta sobre elementos fragmentados não descartados na etapa local. Gera, finalmente, a resposta final que pode ou não ser um mapa. Se a operação envolve a recuperação de um mapa ou parte dele, a etapa de composição ainda é responsável por retirar as fronteiras artificiais e recompor os elementos fragmentados.

Cada uma dessas etapas, contudo, pode ser refinada dependendo do grau de complexidade da consulta, sendo possível a aplicação de filtros iniciais com um plano de execução cuja estrutura equivalenteà definida acima ou operadores locais que requisitem informações sobre fragmentos vizinhos, entre outras variações. O resultado final tipicamente será um plano de execução mais complicado do que o descrito por Q.

A seguir, descreve-se dois exemplos simples para o desmembramento de consultas em dois possíveis planos de execução. Ambos são executados sobre o mapa indicado na figura 5.2.

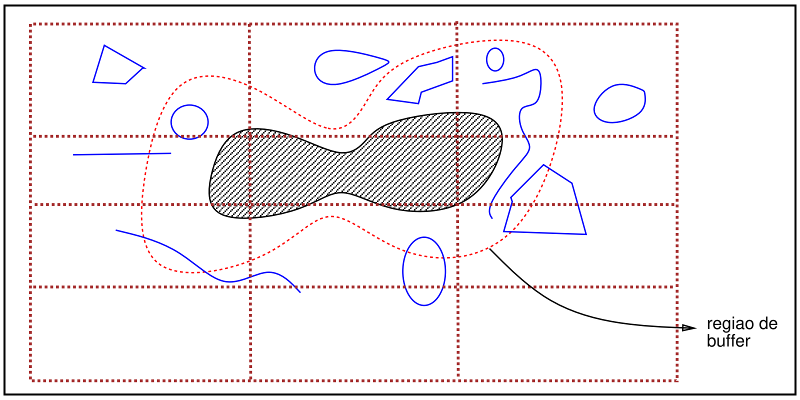

Figura 5.2: Fragmentos do mapa M. Região hachurada é fazenda f.

- Q: *Calcule a área da fazenda* f *representada no mapa* M.
  - $P_Q = D; (A_1//A_2//...A_n); S$
    - D: *broadcast* da operação de cálculo de área de regiões para todos os processadores que cobrem a região do mapa M (segundo catálogo global).
    - A_i: cálculo da área dos fragmentos da fazenda f no processador i.
    - S: composição da resposta dada pela soma das áreas parciais.
- Q: *Retorne os elementos a* x *de distância da fazenda* f.
  - $P_Q = B; V$
    - $B \rightarrow D_b; (B_1//...//B_n); C_b$
    - $V \rightarrow D_v; (V_1//...//V_n); C_v$
  - B: gera *buffer* a x distância da fazenda F.

    * D_b: *broadcast* do cálculo do *buffer* para processadores que cobrem a região do mapa M.
    * B_i: calcula região de *buffer* r_i do fragmento local da fazenda.
    * C_b: compõe as partes da região $r = \bigcup r_i$, que constitui o *buffer* de f.
  - V: seleciona objetos no interior da região gerada em B.
    * D_v: *broadcast* para processadores que cobrem a região r indicando seleção.
    * V_i: retorna fragmento de M induzido por r_i.
    * C_v: gera mapa resposta.

Como pode ser observado, o primeiro exemplo, bastante simples, particiona o processamento da função (cálculo de área) a partir da estrutura do método de armazenamento e compõe, no final, as diversas respostas parciais, somando-as. Já o segundo exemplo, é executado em duas etapas. A primeira delas, que atende a seqüencia *distribuição - execução - composição*, gera a área de interesse a partir do mapa inicial. A segunda parte corresponde a uma seleção espacial a partir de uma janela genérica (*buffer* do polígono f original).

No caso geral, pode-se considerar que as operações locais, dependendo de sua semântica, podem ser: (1) idênticas à equivalente centralizada; (2) não descartam elementos fragmentados; ou (3) não descartam alguns dos elementos fragmentados. Exemplos de cada um deles:

1. *"cálcule a área desmatada em 1990"*. A operação local calcularia a área de cada uma das regiões do mapa e em seguida as somaria. Neste caso, a área desmatada é a soma das áreas das regiões em cada segmento.

2. *"retorne as regiões com área maior que x"*. No caso geral, onde podem existir elementos fragmentados, uma região que em um dos fragmentos não atende à restrição da consulta, pode fazê-lo ao ser composto com as outras partes da mesma região armazenados em nós distintos do sistema. Neste caso, a operação local aplica a restrição normalmente aos elementos inteiros do fragmento, mas não descarta elementos fragmentados da resposta, deixando a análise sobre eles para a etapa de composição da resposta;

3. Uma cláusula muito parecida com o exemplo anterior:*"retorne as regiões* **menores** *que x"* não recai na mesma solução, já que alguns objetos fragmentados

podem ser descartados da resposta ainda na execução da operação local (os que os fragmentos já são maiores que x), mas ainda é necessário um processamento adicional na etapa de composição para checar se os elementos fragmentados não descartados atendem à consulta quando compostos.

O plano de execução pode depender, também da estrutura de distribuição do método usado. Ou seja, situações distintas ocorrem se existe ou não superposição de áreas cobertas por fragmentos distintos. Estruturas que permitem superposição de fragmentos praticamente obrigam uma etapa de composição mais complexa, pois, por exemplo, elementos que se cruzam podem estar em processadores distintos e assim devem ser transferidos para um processador único antes da execução do operador local.

As duas seções a seguir descrevem mais detalhadamente a estratégia a ser seguida na resolução de operações com, respectivamente, um ou mais argumentos. Novamente, é importante lembrar que não é apresentado um procedimento geral para a síntese dos algoritmos já que o objetivo não é a implementação de um interpretador de consultas, mas a ilustração do uso dos métodos de armazenamento propostos na resolução de operações espaciais.

5.3 Operações Geográficas com um Argumento

Formalmente essa classe de operações pode ser definida por $f_\theta(M) = M'$ ou $f_\theta(M) = v$, onde M e M' são mapas, v é um valor escalar e θ é um predicado espacial unário sobre os elementos de M. Um exemplo típico de operação espacial com apenas um argumento é a criação do mapa M' a partir da seleção de objetos num outro dado mapa.

O primeiro padrão de consulta a ser discutido é a seleção do fragmento de mapa induzido por uma janela retangular. Este é o modo mais simples de recuperação de dados espaciais e também o mais comum, utilizado, por exemplo, durante a visualização de uma área de interesse.

O processamento desta seleção resume-se à execução da rotina de recuperação de fragmentos oferecida pelo método de armazenamento e descrita na seção 4.2.3.1, quando é usado o método baseado em grade ou rotina equivalente para cada uma das estruturas de dados possíveis. Essa rotina é estruturada segundo o esquema geral apresentado em 5.2

$$(P_Q = D_j; (G_{j1}//G_{j2}//...G_{jn}); C_j)$$

e seu pseudo-código está repetido na figura 5.3. Ou seja, possui uma etapa inicial de distribuição (D) que identifica os processadores envolvidos e lhes envia um comando get. No método baseado em grade, identifica-se as células de G que estão contidas ou interceptam a região de interesse. Os processadores envolvidos são dados pelo atributo P_c de cada uma das células. Do mesmo modo, no método baseado em árvore R, os processadores são os definidos nos pares (P, B) das folhas que interceptam ou estão contidas na região r. Em seguida, uma etapa de execução constituída pela rotina de recuperação de dados disponibilizada pelo método de acesso local (G_j). Finalmente, a etapa de composição, além de unir os diversos fragmentos recuperados em cada processador, retira as fronteiras artificiais, preparando a resposta para visualização (C_j).

Uma seleção mais genérica (figura 5.4) pode ser feita a partir da indução de um fragmento de mapa por uma região r qualquer. Uma das formas de implementar essa consulta mantém a estrutura do algoritmo anterior:

$$P_Q = D_r; [(F_{j1}; G_{r1})//(F_{j2}; G_{r2})//...(F_{jn}; G_{rn})]; C_r$$

A etapa de distribuição, em primeiro lugar, calcula o retângulo envolvente da região r a partir do qual descobre, da mesma forma que no exemplo anterior, os processadores que cobrem a região especificada (fazendo uso da grade G ou outra estrutura de distribuição, conforme o método de armazenamento usado).

A etapa de execução passa a ser mais complexa e recupera objetos em duas etapas. A primeira (F_{ji}) recupera pelo método de acesso local o fragmento de mapa induzido pelo retângulo envolvente do fragmento de r no processador[1]. O segundo passo da etapa de execução (G_r) trata da geometria dos objetos selecionados, recortando o primeiro fragmento obtido com a região r genérica. Só serão computados como parte da resposta os fragmentos de elementos *no interior* da região r.

Finalmente, a etapa de composição é idêntica à anterior, compondo um mapa-resposta final a partir da união (com desfragmentação) das diversas sub-respostas

[1] Note a diferença do retângulo envolvente do fragmento de r em P e do fragmento em P do retângulo envolvente de r. Na prática qualquer um dos filtros obtém ao final a mesma resposta. O primeiro, porém, restringe mais o domínio de consulta logo na primeira etapa.

Acesso (M, G, p_A, p_B)

Parâmetros:
M = Mapa a ser recuperado.
G = Diretório da grade de distribuição.
p_A, p_B = Pontos de definição da área de interesse.

Algoritmo:

/* DISTRIBUIÇÃO */
ache x_i tal que $x_i \leq x_a$ e não existe x tal que $x_i < x \leq x_a$
ache x_k tal que $x_k \geq x_b$ e não existe x tal que $x_b < x \leq x_k$
ache y_j tal que $y_j \leq y_a$ e não existe y tal que $y_j < y \leq y_a$
ache y_l tal que $y_l \geq y_b$ e não existe y tal que $y_b < y \leq y_l$
para toda célula $c : G[x, y]$ tal que $x_i \leq x < x_k$ e $y_j \leq y < y_l$ /* x e y são índices do diretório G */

/* EXECUÇÃO REMOTA */
    send(P_c,get,(x, y), (next(x), next(y))

/* COMPOSIÇÃO */
Mapa = vazio
para toda célula $c : G[x, y]$ tal que $x_i \leq x < x_k$ e $y_j \leq y < y_l$ e not fim
    receive(P_c, read, aux)
    Mapa = Mapa $\cup_{\oplus}$ aux
retorna Mapa

Comentários Adicionais

next(v) $\rightarrow$ retorna o índice do diretório imediatamente acima do índice v.
A operação receive não bloqueia o *loop* e este é executado até que o processo tenha recebido
    o conteúdo de todas as células (fim = true).
O operador $\cup_{\oplus}$ implementa a operação de desfragmentação definida na seção 3.1.2.

Figura 5.3: Pseudo-código da rotina de recuperação de fragmentos de mapa no método baseado em grade.

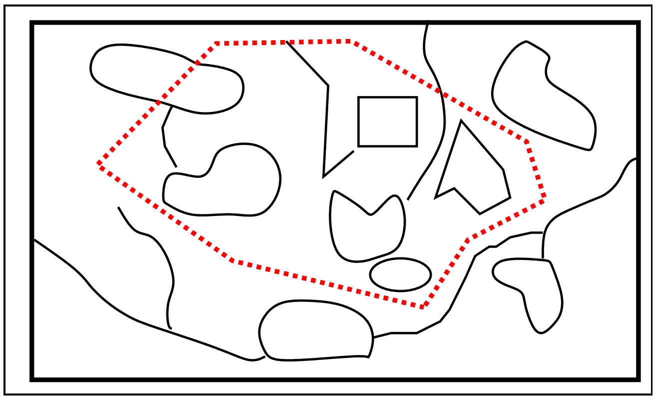

Figura 5.4: Seleção a partir de uma região qualquer, indicada pelo pontilhado.

(C_r).

Note que ambos os casos analisados acima são consultas que geram um mapa desfragmentado como resposta, que pode ser usado pelo módulo de apresentação da aplicação diretamente sem que esse se preocupe com a distribuição do método de armazenamento. Uma outra possibilidade seria a criação de um novo mapa a partir do fragmento recuperado. Neste caso, a etapa de execução local armazena o fragmento resposta segundo o procedimento de criação de mapas.

Um grupo de seleções espacias também considerado é o que engloba consultas que misturam cláusulas espaciais e convencionais. Por exemplo, "*retorne as regiões de solo arenoso no Estado do Rio de Janeiro*". Provavelmente esta consulta será resolvida em dois passos: (1) seleção espacial que segue o procedimento descrito anteriormente; (2) identificação das áreas arenosas entre os objetos selecionados. A separação dos passos pode definir a execução de duas consultas distintas ou pode ser inserida na etapa de execução onde cada processador local retorna respostas parciais que já atendem a ambas as restrições. A vantagem desta última opção se dá pelo aproveitamento da execução em paralelo também para atributos não espaciais. Dependendo da definição dos dados, atributos temáticos (como tipo de solo) podem estar armazenados diretamente na estrutura que implementa as representações espaciais ou no SGBD convencional. Em qualquer um dos casos, a restrição pode ser aplicada pelo processo local com ou sem necessidade de acesso ao SGBD convencional.

A seleção acima, então, pode ser resolvida a partir do seguinte plano de execução:

$$P_Q = D; [(F;T;G)_1 // (F;T;G)_2 // ...(F;T;G)_n]; C$$

onde D e C são equivalentes às etapas de distribuição e composição do exemplo anterior e a seqüencia $(F;T;G)_i$ executa, no processador i, a seleção de filtro a partir do retângulo envolvente do fragmento da região r_j local ao nó, seguido da restrição por tema (são descartadas as regiões não arenosas) e finalmente a seleção pela geometria exata das regiões no interior do fragmento de interesse.

Perceba que a existência de elementos fragmentados ou a superposição da área de cobertura do fragmento em processadores distintos quase não afeta a execução de nenhum dos algoritmos descritos até aqui nesta seção. As diferentes partes de um elemento fragmentado que atende à restrição da consulta são retornados respectivamente pelos diversos nós que os armazenam. A etapa de composição preocupa-se somente em reunir as partes, gerando o elemento inteiro se necessário (operador desfragmentação). O mesmo ocorre no caso de superposição de fragmentos em nós distintos. Cada um deles retorna uma resposta parcial que não altera as demais sub-respostas.

Essa situação, contudo, não ocorre em qualquer consulta. Retornando aos exemplos citados na seção anterior, descreve-se o procedimento que retorna, por exemplo, a área total das regiões maiores que x hectares.

A etapa de distribuição é idêntica às anteriores, identificando os processadores responsáveis pelo tratamento da região de interesse. Na execução local, são calculadas as áreas de todas as regiões armazenadas. Podem ser encontradas três situações:

1. r é uma região inteira e a sua área $A(r)$ é maior que x. Esse valor é, então, totalizado na soma parcial.
2. r é uma região inteira e a sua área $A(r)$ é menor que x. Esse valor é descartado.
3. A região r é um elemento fragmentado. Sua área deve ser retornada associada a seu identificador para ser composta com suas outras partes mais adiante.

A etapa de composição, além de somar as respostas parciais, soma as áreas (já calculadas!) de fragmentos de cada um dos elementos não descartados. Caso a área resultante seja maior que x, o valor é somado à área total, senão é descartado.

5.4 Operações Geográficas com mais de um Argumento

A classe de operações definida $f_\theta(M_0, M_1, ..., M_n) = R$ caracteriza-se por relacionar um número de mapas maior que dois, a partir de um atributo espacial. Exemplos típicos são as junções espaciais determinadas pela interseção, cruzamento, pertinência ou proximidade de objetos em vários mapas e a operação de *map overlay*. Tratamos aqui apenas das operações de junção que envolvem dois mapas, dado por $f_\theta(M, M') = R$, mas todas as considerações feitas podem ser facilmente expandidas para o caso geral.

Uma junção é definida pela combinação dos elementos de dois conjuntos de dados num único conjunto de pares tais que cada par de elementos oriundos das duas estruturas iniciais satisfazem uma dada condição [HS94]. A junção é dita espacial se a condição que a define é um relacionamento espacial. Os tipos mais comuns [BKS93] retornam pares de objetos cujos retângulos envolventes se interceptam ou um novo elemento: a própria interseção dos elementos.

Na verdade, essa operação é determinada pela composição de um mapa R a partir de fragmentos de mapas gerados pela aplicação do operador f_θ local sobre os fragmentos dos mapas M e M' em cada processador. A vantagem do uso do método de resolução apresentado neste texto é fazer uso de algoritmos já existentes para os métodos de armazenamento local e ainda minimizar os gargalos de eficiência já que fragmenta e distribui o processamento por diversos processadores. Em outras palavras, o método apresentado não deve substituir algoritmos de junção já estudados, mas complementa-los usando a política de fragmentação dos mapas envolvidos. Algoritmos eficientes para o processamento de junções espaciais podem ser encontradas na literatura, por exemplo em [BKS93] no caso da estrutura de armazenamento usada ser R^**-trees*, ou em [HS94] para *PMR-quadtrees*. Esse último faz uso, inclusive, de processamento paralelo.

O primeiro padrão de junção analisada retorna pares de elementos dos mapas originais, a partir da interseção de seus retângulos envolventes. Essa operação pode ser usada para se obter resultados aproximados mais depressa ou como filtro inicial na resolução de junções sobre a geometria exata de elementos. O plano de execução dessa operação é dado por:

$$P_Q = D; (P_1//P_2//...P_n); C$$

onde a etapa de **distribuição** D identifica os processadores envolvidos. Os processos P_i executam localmente a junção com o algoritmo mais apropriado ao método de acesso local. Em seguida, a etapa de **composição** une as respostas parciais, eliminando pares repetidos oriundos de possíveis elementos fragmentados que interceptam o mesmo elemento em diferentes fragmentos.

O procedimento acima só é válido, porém, quando a estrutura de distribuição do método de armazenamento não permite superposição de áreas cobertas por fragmentos. A operação acima apresenta um plano de execução bem mais complexo, quando implementada sobre o método de armazenamento baseado em árvores balanceadas. Por exemplo:

$$\begin{aligned} P_Q &= (P_1//P_2); C \\ P_1 &= D_P; J_1//J_2//...J_n \\ P_2 &= D_\cap; (T_1; K_1)//(T_2; K_2)//...(T_m; K_m) \end{aligned} \tag{5.1}$$

A resolução da operação fica, então, dividida em dois módulos executados em paralelo. O primeiro deles, P_1, identifica os nós envolvidos (D_P) e executa a junção local normalmente (J_i). Ao mesmo tempo, o procedimento P_2 identifica áreas de interseção entre as folhas da árvore de distribuição que estejam na região de interesse da consulta ($D_\cap$). Para cada uma delas, é feita uma seleção dos elementos de um dos fragmentos contidos na região de interseção dos nós que é transferido para o outro nó (T_i). Neste, é executada uma variação da junção local (K_i) que testa a interseção dos retângulos envolventes entre elementos de nós distintos. Essa variação do operador de junção é usada para evitar que sejam testados relacionamentos processados nas etapas J_i.

Heurísticas que tentem prever a disponibilidade de tempo de processamento nos nós envolvidos devem ser usadas na escolha do nó onde será executado o procedimento K_i. De preferência, deve ser usado um dos processadores envolvidos (que armazenam um dos fragmentos que se interceptam), mas a transferência de ambos os fragmentos para um terceiro processador ocioso também pode ser feita.

Finalmente, o processo C compõe as sub-respostas. A figura 5.5 ilustra a junção de dois mapas $M1$ e $M2$ armazenados em dois processadores.

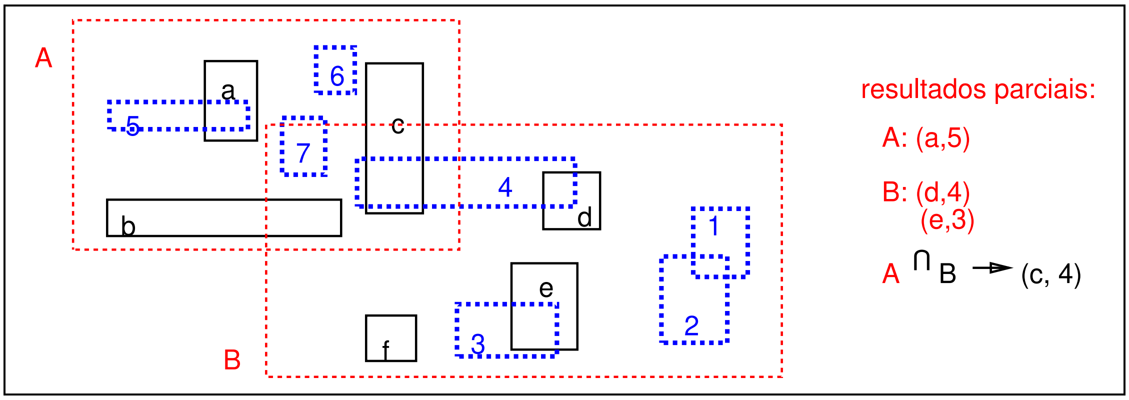

Figura 5.5: Respostas parciais da junção que retorna os elementos de $M1$ e $M2$ cujos retângulos envolventes se interseptam.

O fato dos dois procedimentos acima dependerem do método de armazenamento usado não significa que não se pode definir um procedimento único aplicável em qualquer estrutura de dados. Note que o plano da equação 5.1 pode ser aplicado no caso anterior. Deste modo, o processo $D_{\cap}$ não identificará nenhuma região de interesse, pois a grade G não permite superposição de células e P_2 não será executado reduzindo o problema ao algoritmo apresentado anteriormente.

O próximo tipo de operação retorna pares de elementos cujas geometrias exatas se interceptam. Alguns exemplos: (1) *"retorne rios e estradas que se cruzam."* - linha cruza linha; (2) *"retorne municípios na fronteira dos estados."* - região toca linha; (3) *"retorne as culturas plantadas nas fazendas."* - região intercepta região. Essas operações são executadas como as do exemplo anterior. Apenas a operação de junção local difere, comparando a geometria dos elementos envolvidos.

Até agora as operações de junção estudadas, selecionam, a partir de um determinado relacionamento, um sub-conjunto do produto cartesiano dos elementos de dois mapas sem que haja uma pré-seleção não espacial. O próximo grupo analisado usa também um atributo dos elementos de pelo menos um dos mapas. Um exemplo é: *"retorne estradas federais que cruzam rios principais"*. A junção definida nesta consulta não testa o cruzamento de qualquer rio com as estradas, mas somente os rios principais e as estradas federais. Neste caso, o otimizador deve optar se será feita uma seleção prévia dos rios principais e das estradas federais, ou de um deles ou selecioná-los depois do uso de um filtro espacial (retângulos envolvente que se interceptam, por exemplo). Novamente, a vantagem do método proposto é deixar que essa decisão fique por conta dos processos locais aos fragmentos que funcionam do mesmo modo que numa arquitetura convencional. Em outras palavras, o plano de execução sugerido para esse grupo de operações é idêntico aos anteriores. Uma melhora possível pode ser feita nas etapas T_j que podem selecionar os elementos pelo atributo convencional diminuindo a quantidade de dados transferidos entre processadores.

O próximo grupo de junções estudadas envolvem distâcia em exemplos do tipo: (1) *" retorne as casas a uma distância* x *da* BR-101*"*; ou *"retorne os pares de postes que estão a até* x *km de distância entre si."*

O primeiro exemplo pode ser reduzido a uma seleção por *buffer*, diferindo do exemplo da seção 5.3 pelo fato do elemento que define o *buffer* não estar no mapa em que é feita a seleção. Seu plano de execução fica, então, definido por:

$$P_Q = B; V$$

$$B = D_b; (B_1//B_2//...B_n); C_b$$

$$V = D_v; (V_1//V_2//...V_n); C_v$$

onde B calcula a região de interesse a partir do elemento do primeiro mapa e V executa uma seleção espacial genérica no segundo mapa a partir da região definida por B.

Esta operação pode ser resolvida por um procedimento mais genérico que resolve também o segundo exemplo. Este procedimento, mais genérico e complexo, é executado da seguinte forma:

$$P_Q = D; (P_1//P_2//...P_n); C$$

$$P_i = B(e); [(V_l(b(e))//V_r(b'(e))]$$

D identifica os processadores envolvidos. Em cada um deles, para cada elemento é calculado seu *buffer* a uma distância x ($B(e)$). Em seguida é feita a seleção que pode ocorrer em dois processos em paralelo. Um deles, V_l, é responsável pela operação de varredura local. Se o *buffer* gerado invadir a área de um outro processador, V_r é executado, enviando uma consulta remota a ser computada pelo nó vizinho.

Finalmente, C compõe as respostas parciais. A figura 5.6 ilustra o teste feito a partir do ponto p. Calculado o *buffer* $B(p)$ são iniciadas duas seleções espaciais V_l, local, indicada pela hachura em diagonal e V_r, remota, indicada pela hachura quadriculada.

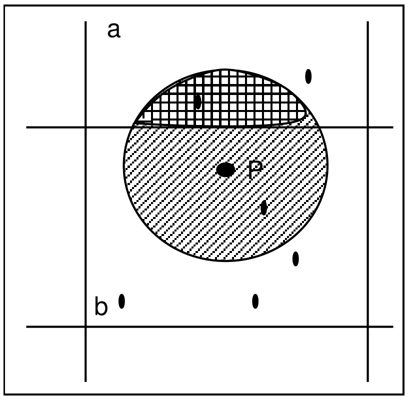

Figura 5.6: Área do *buffer* do ponto P.

A última variação de junção possível é a mais complexa por retornar como resposta não um par de elementos existentes, mas um novo elemento gerado a partir da junção. É o caso, por exemplo, de *"retorne as áreas de clima temperado e solo arenoso"*. O objetivo desta operação é identificar as interseções existentes e não os pares que se interceptam.

Novamente o plano de execução é dado, no caso geral, pela expressão:

$$P_Q = (P_1//P2); C$$

$$P_1 = D_p; (J_1//J_2//...J_n)$$

$$P_2 = D_\cap; [(T_1; K_1)//(T_2; K_2)//...(T_m; K_m)]$$

Existem duas diferenças básicas deste procedimento para o já estudado: (1) a rotina de junção local retorna elementos e elementos fragmentados gerados a partir da interseção entre elementos dos mapas originais. A geração de um elemento fragmentado ocorre a partir da interseção entre dois elementos em que pelo menos um é fragmentado; (2) a etapa de composição C deve, ao retirar as fronteiras artificiais, juntar partes de elementos fragmentados retornados nas respostas parciais. Note que se o fragmento de um elemento não encontra seu outro fragmento, ele configura um elemento inteiro. A resposta final dessa operação tanto pode ser um conjunto de regiões, como pode gerar um novo mapa armazenado através da rotina de criação de mapas do método de armazenamento.

Operações de *map overlay* são muito parecidas com esse último exemplo quando não há seleção por atributo envolvida. Ou seja, computa-se, por exemplo, o mapa de potencial agrícola através da interseção entre os mapas de clima e tipo de solo sem descartar nenhum par encontrado. É possível, ainda, expandir a etapa de composição para mapear pares de temas num tema apropriado, por exemplo, clima $x_i \cap$ solo y_j é mapeado para cultura w_k, para qualquer i, j, k.

Capítulo 6

Considerações Finais

Neste capítulo, faz-se as considerações finais sobre o trabalho desenvolvido. A seção 6.1 faz um breve resumo da dissertação, a seção 6.2 comenta as principais contribuições feitas e finalmente, a seção 6.3 propõe algumas extensões a serem desenvolvidas a partir deste trabalho.

6.1 Resumo

Partindo de um modelo e de uma arquitetura genérica para a gerência de objetos geográficos, este trabalho enfatizou o tratamento desses objetos através de representações em forma de mapas vetoriais, concentrando-se nas questões de armazenamento e resolução de operações espaciais. Introduziu-se um método de armazenamento distribuído que tem como objetivo dar suporte à execução paralela de operadores espaciais, minimizando o custo de certas operações geográficas.

Foi considerado o uso de uma arquitetura paralela segundo um modelo de memória distribuída que pode ser uma máquina paralela ou um *cluster* de estações de trabalho ligadas por um meio de comunicação de alto desempenho. Neste ambiente, cada nó é independente e conhece a semântica inerente a mapas vetoriais e operações espaciais.

O método proposto, baseado em dois níveis - distribuição e armazenamento - é genérico e facilmente estendido para suportar diferentes estruturas de dados em qualquer um dos níveis. A dissertação apresentou em detalhes uma das formas possíveis, baseada em grades, mas também fez importantes considerações a cerca da generalização da estrutura.

Finalmente, discorreu-se sobre a resolução de operações geográficas de dois tipos, a recuperação de fragmentos de mapas e junções por atributos espaciais. O método proposto faz uso, para incrementar a eficiência de consultas, de conceitos de paralelismo e fragmentação de objetos. Permite que diversos nós cooperem com a resolução das cláusulas espaciais das consultas em um SIG e faz uso de métodos de acesso e algoritmos pré-existentes.

6.2 Contribuições

As principais contribuições da dissertação centram-se na definição de uma família de métodos de armazenamento de mapas vetoriais. Ela é baseada na distribuição de fragmentos de mapas induzidos por células de fragmentação. Esse método oferece uma base para a implementação distribuída do sub-sistema de armazenamento vetorial de gerenciadores de objetos geográficos.

Entre outras considerações, destaca-se o conceito de fragmentação de mapas e também de elementos geográficos que permite a cooperação de diferentes processadores na execução de operadores complexos.

O trabalho introduziu, também, um esquema geral para o processamento distribuído de operações espaciais tomando como base a estrutura proposta. Foram abordadas duas classes de operações: as que envolvem apenas a recuperação de fragmentos de um único mapa e as que envolvem fragmentos de mais de um mapa, como junções por atributos espaciais e *map overlay*.

6.3 Possíveis extensões

Uma primeira extensão sugerida é o detalhamento de variações do método proposto com o uso de estruturas de distribuição mais complexas. Pode ser feita, ainda, uma análise quantitativa que implemente e comprove a eficiência das diferentes variações do método proposto e auxilie na definição de parâmetros adequados para seu uso (número de processadores, tamanho dos fragmentos, estrutura de armazenamento, etc...)

Aspectos de balanceamento de carga ainda podem ser bastante explorados, incrementando as heurísticas usadas para a distribuição dos fragmentos pelos nós do sistema.

Dentro do contexto do processamento de operações espaciais, o esquema geral proposto pode ser detalhado para a definição de um interpretador e/ou otimizador de consultas que gere código otimizado para as estruturas de armazenamento distribuído.

Outro ponto interessante seria a extensão do método para dados não vetoriais (mais especificamente, imagens ou dados matriciais) que também possuem algoritmos de manipulação pesados e que poderiam fazer uso de paralelismo a partir de uma estrutura de distribuição semelhante a aqui proposta.

Referências Bibliográficas

[Bar93] Valmir C. Barbosa. **Massive Parallel Models of Computation**. Ellis Horwood, 1993.

[Ben75] Jon Louis Bentley. **Multidimensional Binary Trees Used for Associative Searching**. *Communications of ACM*, 18(9), September 1975.

[BER85] D.A. Beckley, M.W. Evens, and V.K. Raman. **Multikey Retrival from K-d trees and Quad-trees**. In *Proceedings of ACM SIGMOD Conference on Management of Data*, May 1985.

[BH+93] Thomas Brinkhoff, , Holger Horn, Hans P. Kriegel, and Ralf Schneider. **A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems**. In *Proceedings of Third International Symposium of Advances in Spatial Databases*, June 1993.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. **Efficient Processing of Spatial Joins Using R-Trees**. In *Proceedings of the 1993 ACM SIGMOD*, volume 22, Washington, DC, June 1993.

[BKSS90] Nobert Beckmann, Hans P. Kriegel, Ralf Schneider, and Benhard Serger. **The R*-tree: an Efficient and Robust Access Method for Points and Rectangles**. In *Proceedings of the 1990 ACM Sigmod International Conference on Management of Data*, 1990. Vol.19, No.2.

[BKSS94] Thomas Brinkhoff, Hans P. Kriegel, Ralf Schneider, and Benhard Serger. **Multi-Step Processing of Spatial Joins**. In *Proceedings of the 1994 ACM Sigmod International Conference on Management of Data*, May 1994.

[CFO93] Eliseo Clementini, Paolino Felice, and Peter Ooesterom. **A Small Set of Formal Topological Relationships Suitable for End-User In-**

terfaces. In *Proceedings of Third International Symposium of Advances in Spatial Databases*, June 1993.

[CFS+94] Gilberto Câmara, Ubirajara Freitas, Ricardo C.M. Souza, Marco A. Casanova, and Andréa S. Hemerly. **Object-Oriented Data Modelling and Interface Design for GIS**. In *International Journal of Geographic Information System*, 1994.

[CGMS94] Nicholas J. Carriero, David Gelernter, Thmothy Mattson, and Andrew H. Sherman. **The Linda Alternative to Message-Passing Systems**. *Parallel Computing*, 20(4), April 1994.

[Cia94] Paolo Ciaccia. **Parallel Independent Grid Files Based on a Dynamic Declustering Method Using Multilpe Error Correcting Codes**. Technical Report 9421, Laboratoty for Computer Science - University of Bologna, November 1994.

[Cif95] Ricardo Rodrigues Ciferri. **Um *benchmark* voltado à análise de desempenho de sistemas de informações geográficas**. Master's thesis, UNICAMP, June 1995.

[CK89] C. H. Chien and T. Kanade. **Distributed Quadtree Processing**. In *Design and Implementation of Large Spatial Databases - SSD*, Santa Barbara, California, July 1989.

[CMM67] Richard W. Conway, William L. Maxwell, and Louis W. Miller. **Theory of Scheduling**. Addison-Wesley Publishing Company, 1967.

[Cox91] Frederico Sidney Cox. **Análise de Métodos de Acesso a Dados Espaciais Aplicados a Sistemas Gerenciadores de Banco de Dados**. Master's thesis, UNICAMP, December 1991.

[DG90] David DeWitt and Jim Gray. **Parallel Database Systems: The Future of Database Processing or a Passing Fad**. *SIGMOD Record*, 19(4), December 1990.

[DG92] David DeWitt and Jim Gray. **Parallel Database Systems: The Future of High Performance Database Systems**. *Communications of the ACM*, 35(6), June 1992.

[DKL+94] David DeWitt, Navin Kabra, Jun Luo, Jignesh Patel, and Jie-Bing Yu. **Client-Server Paradise**. In *Proceedings of the 20th VLDB Conference*, 1994.

[DLPY93] David DeWitt, Jun Luo, Jignesh Patel, and Jie-Bing Yu. **Paradise - a Parallel Geographic Information System**. In *Proceedings of the ACM Workshop on Advances in GIS*, 1993.

[Dow93] Kevin Dowd. **High Performance Computing**. O'Reilly & Associates, Inc., 1993.

[EF91] Max Egenhofer and Robert Franzosa. **Point-set Topological Spatial Relations**. *International Journal of GIS*, 5(2), 1991.

[ELS95] Georgios Evangelidis, David Lomet, and Betty Salzberg. **The hB$^{\Pi}$-tree: A Modified hB-tree Supporting Concurrency, Recovery and Node Consolidation**, 1995. Accepted to VLDB'95 Conference.

[ES93] Georgios Evangelidis and Betty Salzberg. **Using the Holey Brick Tree for Spatial Data in General Purpose DBMSs**. *Bulletin of the Technical Committe on Data Engineering*, 16(3), September 1993.

[FK93] Christos Faloutsos and Ibrahim Kamel. **High Preformance R-tree**. *Bulletin of the Technical Committe on Data Engineering*, 16(3), September 1993.

[GDQ92] Shahram Ghandeharizadeh, David DeWitt, and Waheed Qureshi. **A Performance Analysis of Alternative Multi-Attribute Declustering Strategies**. *ACM SIGMOD*, 21(2), June 1992.

[Goo91] M. Goodchild. **Integrating GIS and Environmental Modeling at Global Scales**. In *Proc GIS/LIS 91*, volume 1, pages 117–127, 1991.

[Goo92] Michael F. Goodrich. **Geographical Data Modeling**. *Computers & Geosciences*, 18(4), 1992.

[Gra93] Goetz Graefe. **Query Evaluation Techniques for Large Databases**. *ACM Computing Surveys*, 25(2), June 1993.

[Gut84] A. Guttman. **R-trees: A Dinamic Index Structure for Spatial Searching**. In *Proceedings of the ACM SIGMOD Conference in Data Engineering*, June 1984.

[Gut94] Ralf Hartmut Guting. **An Introduction to Spatial Database Systems**. *VLDB Journal, Spatial Database Systems*, 3(4), October 1994.

[Her92] John Herring. **Tigris: a Data Model for an Object-Oriented GIS**. *Computer & Geosciences*, 18(4), 1992.

[HMS+94] Andréa S. Hemerly, Maurício Mediano, Paulo Souza, Claudia A. Tocantins, and Marcelo Duarte. **Uma Arquitetura para um Gerenciador de Objetos Geográficos**. Documento Interno - Centro Científico - Rio, 1994.

[HS78] Ellis Horowitz and Sahni. **Fundamentals of Computer Algorithms**. Computer Science Press, 1978.

[HS92] Erik G. Hoel and Hanan Samet. **A Qualitative Comparison Study of Data Structures for Large Line Segments**. In *Proceedings of ACM SIGMOD Conference*, 1992.

[HS94] Erik G. Hoel and Hanan Samet. **Performance of Data-Parallel Spatial Operations**. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[INP94] INPE, editor. **Spring - Manual de Teoria**. Instituto Nacional de Pesquisas Espaciais, April 1994.

[ITS94] ITSO, editor. **Introduction to Parallel Processing and Scalable POWERparallel Systems 9076 SP1 and 9076 SP2**. International Technical Support Organizations - IBM, May 1994.

[KF92] Ibrahim Kamel and Christos Faloutsos. **Parallel R-Trees**. *ACM SIGMOD Record*, 21(2), June 1992.

[LDSY91] R. A. Lorie, J. Daudenarde, J.W. Stamos, and H. C. Young. **Exploiting Database Paralelism in a Message-Passing Multiprocessor**. *IBM Journal of Research and Development*, 35(5/6), September 1991.

[LO93] Hongjun Lu and Beng-Chion Ooi. **Spatial Indexing: Past and Future**. *Bulletin of the Technical Committe on Data Engineering*, 16(3), September 1993.

[Mat93] Marta Lima Queirós Mattoso. **Aspectos de Paralelismo na Gerência de Dados e Objetos no GeoTaba**. PhD thesis, COPPE - Universidade Federal do Rio de Janeiro, April 1993.

[MCD94] Maurício Mediano, Marco A. Casanova, and Marcelo Dreux. **V-Trees - A Storage Method for Long Vector Data**. In *Proceedings on 20th VLDB Conference*, September 1994.

[Med95] Maurício Riguette Mediano. **V-trees: Um Método de Armazenamento para Objetos Vetoriais Longos**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, January 1995.

[Mor92] Scott Morehouse. **The Arc/Info GIS**. *Computers & Geosciences*, 18(4), 1992.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. **The Grid file: An Adaptable, Symmetric Multikey File Structure**. *ACM Transactions on Database systems*, 9(1), March 1984.

[OMSD87] Beng Chin Ooi, Ken J. McDonell, and Ron Sacks-Davis. **Spatial kd-tree: an Indexing Mechanism for Spatial Databases**. In *Proceedings of IEEE Comp. Software Applications Conference*, 1987.

[OV91] M. Tamer Ozsu and Patrick Valduriez. **Principles of Distributed Database systems**. Prentice-Hall International Editions, 1991.

[PD95] Enrico Puppo and Giuliana Dettori. **Towards a Formal Model for Multiresolution Spatial Maps**, 1995. Accepted to the Fourth International of Symposium of Large Spatial Databases - SSD'95.

[Rez92] Marco A. M. Rezende. **Métodos de Acesso para Bancos de Dados Espaciais**. Technical Report CCR-142, Centro Científico Rio, January 1992.

[RL85] Nick Roussopoulos and Daniel Leifker. **Direct Spatial Search on Pictorial Databases Using Packed R-trees.** In *Proceedings of the ACM SIGMOD Conference on Management os Data*, May 1985.

[RM92] Jonathan Raper and David Maguire. **Design Models and Funcionality in GIS**. *Computers & Geosciences*, 18(4), 1992.

[RS94] Pedro J. Rezende and Jorge Stolfi. **Fundamentos de Geometria Computacional**. IX Escola de Computação, Recife, July 1994.

[Sam90] Hanan Samet. **The Design and Analysis of Spatial Data Structures**. Addison-Wesley, 1990.

[Sch90] Donovan Schneider. **Complex Query Processing in Multiprocessor Database Machines**. PhD thesis, University of Wisconsin-Madison, September 1990.

[SGRM94] V.S. Sunderam, G.A. Geist, R.Dongarra, and R. Manchek. **The PVM Concurrent Computing Sysmtem: Evolution, Experiences and Trends**. *Parallel Computing*, 20(4), April 1994.

[SK91] Ralf Schneider and Hans-Peter Krigel. **The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations**. In *Proceedings of 7th Workshop on Computational Geometry*, 1991.

[Sny95] Norm Snyder. **IBM POWER Parallel Systems - Parallel Primer**. IBM POWER Parallel Systems, 1995.

[SR87] M. Stonebracker and L. Rowe. **The Postgres Data Model**. In *XIII Very Large Database Conference*, Bringhton, England, 1987.

[SRF87] Timos Sellis, Nick Roussopoulus, and Christos Faloutsos. **The R$^+$-Tree: A Dynamic Index for Multi-Dimensional Objects**. In *Proceedings of the 13th VLDB Conference*, September 1987.

[SSA$^+$94] Craig B. Stunkel, Dennis G. Shea, Bullet Abali, Mark Atkings, Carl A. Bender, Don G. Grice, Peter H. Hochschild, Douglas J. Joseph, Ben J. Nathanson, Richard A. Swetz, Robert F. Stucke, Michael Tsao, and Philip R. Varker. **The SP2 Communication Subsystem**. IBM internal paper, 1994.

[Sto86] Michael Stonebracker. **The Case for Shared-Nothing**. *IEEE Database Engineering*, 1(9), March 1986.

[Toc93] Claudia Andrade Tocantins. **Um Estudo de Estruturas de Dados Topológicos Aplicado a um Sistema de Informações Geográfica**. Estudo Orientado PUC-Rio, 1993.

[Wan93] Fangju Wang. **A Parallel Intersection Algorithm for Vector Polygon Overlay**. *IEEE Computer Graphics & Applications*, 1993.

[WDYT91] Joel L. Wolf, Daniel M. Dias, Philip S. Yu, and John Turek. **Comparative Performance of Parallel Join Algorithms**. Technical Report RC17035, York Town Heights, IBM Research Division, March 1991.

[WH92] Thomas C. Waugh and Sara Hopkins. **An algorithm for polygon overlay using cooperative processing**. *International Journal Geographical Information Systems*, 6(6), 1992.

[WHM90] M. Worboys, H. Hearnshaw, and D. Maguire. **Object-Oriented Data Modeling for Spatial Databases**. *International Journal of Geographical Information Systems*, 4(4), 1990.

ARMAZENAMENTO E MANIPULAÇÃO DE MAPAS VETORIAS EM UMA ARQUITETURA PARALELA

Dissertação de Mestrado apresentada por **Claudia de Andrade Tocantins** em 30 de agosto de 1995 ao Departamento de Informática da PUC-RJ, e aprovada pela Comissão Julgadora formada pelos seguintes professores:

Prof. Marco Antônio Casanova (Orientador)
Centro Científico Rio — IBM Brasil

Prof^{a}. Noemi Rodriguez
Departamento de Informática – PUC-Rio

Prof. Sérgio Lifschitz
Departamento de Informática – PUC-Rio

Visto e permitida a impressão.
Rio de Janeiro, 30 de agosto de 1995.

Coordenador de Programas de Pós-Graduação do
CENTRO TÉCNICO CIENTÍFICO