Explorando Conceitos e Mecanismos de Memória Compartilhada Distribuída em Entrada/Saída Paralela*

Carla Osthoff [1], Ricardo Bianchini [2], Cristiana Seidel [3], Marta Mattoso [2] e Claudio L. Amorim [2]

[1] Departamento de Computação Eletrônica - LNCC
[2] Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ
[3] Departamento de Engenharia de Sistemas e Computação - UERJ
osthoff@lncc.br, {ricardo,marta,amorim}@cos.ufrj.br, seidel@eng.uerj.br

Abstract—
Parallel applications from several areas, such as scientific computing and commercial databases, require high-performance input/output (I/O) systems. This paper proposes the exploitation of software-based distributed shared-memory (software DSM) concepts and mechanisms to optimize disk caching and, as a result, substantially improve the I/O performance of parallel systems. More specifically, the main contribution of the paper is a set of mechanisms that allow us: (a) to move the coherence of disk data to the main memory level; (b) to utilize a relaxed consistency model for the disk data accesses; and (c) to save disk cache space. In order to evaluate our ideas, we are currently implementing the DSMIO system for a prototype parallel database manager using the IBM-SP multicomputer system. Our preliminary results show that the database benchmarks that benefit the most from our system can achieve 99% reductions in execution time. We conclude that the exploitation of software DSM concepts and mechanisms indeed significantly improve the I/O performance of parallel database applications.

Keywords— **Memória Compartilhada Distribuída, Entrada/Saída Paralela, Sistemas de Banco de Dados Distribuídos**

I. Introdução

Sistemas para a movimentação de dados entre a memória primária e os dispositivos externos, chamados comumente de sistemas de entrada e saída (E/S), são parte importante de qualquer plataforma computacional. De fato, um alto desempenho do sistema de E/S é crucial para aplicações de diversas áreas, tais como computação científica e bancos de dados. No entanto, a pesquisa em sistemas de E/S mostra que tem sido difícil atingir esse alto desempenho, especialmente para aplicações paralelas.

Em vista disso, um grande número de pesquisas tem sido realizadas na tentativa de reduzir a latência e aumentar o *throughput* da E/S [7, 1]. Apesar de todos os avanços nessa área, o tempo médio de acesso aos dados de disco ainda domina a execução de um grande conjunto de aplicações paralelas. Para tal conjunto, novas otimizações provavelmente devem vir de uma melhor utilização da memória principal para *caching* dos dados de disco, já que o tempo médio de acesso à memória é várias ordens de grandeza menor do que o tempo médio de acesso à memória secundária.

Assim sendo, este artigo explora os conceitos e mecanismos encontrados nos sistemas de memória compartilhada distribuída (Distributed Shared Memory, DSM) baseada em *software* para otimizar o cacheamento dos dados de disco, de forma a aumentar substancialmente o desempenho da E/S em sistemas paralelos.

Nossa proposta vem da observação de que sistemas DSM e sistemas de arquivos paralelos e/ou distribuídos têm formas semelhantes de abstrair a localização dos dados acessados. Mais especificamente, sistemas DSM provêem a abstração de uma memória compartilhada em um ambiente distribuído, onde as memórias distribuídas são mapeadas num espaço de endereçamento único de forma transparente à aplicação. Um sistema de arquivos paralelo e/ou distribuído fornece uma abstração análoga no nível da memória secundária, uma vez que dados armazenados em qualquer disco podem ser acessados transparentemente pela aplicação. Em um sistema de *cache* de disco tradicional, cada servidor de arquivo possui uma *cache* independente para armazenar os dados do disco. Neste esquema, quando há compartilhamento, a coerência dos dados só é garantida no próprio disco [13] e o desempenho de aplicações com altas taxas de escrita de dado se torna desastroso [17].

Podemos evitar esse tipo de problema estendendo os conceitos e mecanismos utilizados em sistemas DSM para garantir a coerência dos dados no nível da memória principal. O foco principal dessa extensão é que todas as *caches* de disco do sistema devem formar uma *cache* única, compartilhada por todos os processadores. Além disso, estamos utilizando um modelo relaxado de consistência de memória no acesso a essa *cache* compartilhada o que faz com que ocorra uma redução na comunicação necessária para manter estas memórias consistentes. Permitimos ainda múltiplos escritores concorrentes aos blocos de disco através do mecanismo de *diffing*, onde apenas as atualizações feitas a um dado pertencente ao bloco são mantidas localmente e não

*Trabalho parcialmente financiado pela Finep e CNPq.

o dado inteiro. Essa última otimização permite que economizemos espaço de armazenamento em memória principal. Neste trabalho não estamos avaliando ainda as possibilidades de economia de memória de DSMIO.

Para avaliar essas idéias, estamos construindo o sistema Distributed Shared Memory I/O, DSMIO para um multicomputador IBM-SP. A idéia básica do sistema DSMIO é integrar o sistema de *cache* de disco a um sistema *software DSM*, no caso o sistema TreadMarks [10]. A integração entre esses dois sistemas, na verdade, pode ser vista por dois ângulos distintos: o sistema DSMIO provê ao sistema TreadMarks a possibilidade de realizar E/S de forma otimizada; ou TreadMarks provê ao sistema DSMIO uma memória compartilhada e mecanismos de coerência que permitem a implementação de uma *cache* compartilhada.

No momento, as nossas aplicações alvo são de processamento de transações e de suporte a decisão. Por esse motivo, estamos testando nosso sistema como uma extensão de um sistema de gerência de banco de dados. Nossos resultados preliminares mostram que o *overhead* introduzido por DSMIO é insignificante. Nossos resultados mostram ainda que DSMIO obtém reduções de tempo de execução de até 99% em relação a um tradicional, através da eliminação da maior parte das escritas em disco.

Baseados nesses resultados, concluímos que o uso de conceitos e mecanismos de sistemas *software DSM* de fato melhoram significativamente o desempenho de E/S de aplicações paralelas. Para o futuro, pretendemos estudar novas classes de aplicações e aumentar a confiabilidade do sistema.

O artigo está organizado da seguinte forma. Na seção seguinte apresentamos alguns conceitos básicos sobre sistemas *software DSM*. Na seção III descrevemos o sistema DSMIO. Na seção IV detalhamos a metodologia utilizada nos nossos experimentos. Na seção V avaliamos nossos resultados. Na seção VI discutimos os trabalhos relacionados. Na seção VII apresentamos nossas conclusões.

II. Sistemas Software DSM

Sistemas DSM com coerência de dados mantida por software têm se mostrado uma alternativa de baixo custo para programação no modelo de memória compartilhada. Esses sistemas normalmente utilizam o *hardware* de memória virtual para manutenção da coerência dos dados compartilhados.

O sistema TreadMarks é o sistema de *software DSM* mais utilizado nos dias de hoje e é o sistema base utilizado por DSMIO. TreadMarks emprega a página como unidade de coerência, utiliza protocolo de invalidação para manutenção da coerência dos dados compartilhados e provê suporte a múltiplos escritores através de mecanismo de *diffing*. Antes de começar a atualizar uma página o sistema cria uma cópia gêmea da mesma, chamada *twin*. As escritas são realizadas livremente na página e o *twin* mantém a versão original. Quando as escritas locais têm que ser propagadas, o processador compara a cópia corrente da página com o seu *twin*, gerando, em uma estrutura chamada *diff*, as diferenças entre eles.

O modelo de consistência empregado por TreadMarks é o *Lazy Release Consistency* (LRC) [9], que atrasa a propagação das informações de coerência até a próxima aquisição de uma sincronização (e.g, *lock*). O sistema divide a execução do programa em intervalos e utiliza *timestamps* para determinar quais modificações no dado compartilhado um processador deve observar na aquisição da sincronização. As informações sobre as modificações realizadas num dado compartilhado são transferidas de um processador para outro na forma de notificações de escrita em uma página, ou *write-notices*.

III. O Sistema DSMIO

Nessa seção apresentamos o sistema DSMIO que ilustra como os mecanismos utilizados em sistemas DSM podem ser estendidos para garantir a coêrencia dos dados no nível da *cache* do disco.

A. Estrutura Geral

O sistema DSMIO utiliza uma área da memória compartilhada estabelecida por TreadMarks, chamada de *cache* compartilhada, para armazenar os dados lidos do disco. Entretanto, as modificações realizadas nos dados são armazenadas na memória local do nó na forma de *diffs* (i.e., o nó armazena apenas a diferença entre o dado original e o dado modificado).

A coerência das páginas da cache compartilhada é mantida segundo o modelo LRC de consistência de memória adotado por TreadMarks. Assim, o sistema DSMIO mantém toda a estrutura de intervalos e *write-notices* utilizada por TreadMarks. Mais especificamente, numa operação de *lock* (entrada de uma seção crítica), o processador recebe os *write-notices* relativos às escritas realizadas nos dados do disco e invalida suas respectivas páginas. Em uma próxima operação de leitura do disco o processador é capaz de recuperar o dado através dos *write-notices* recebidos.

Para que um nó possa localizar a versão original do dado na *cache* compartilhada, o sistema DSMIO estabelece estaticamente um nó *home* para cada dado lido do disco. Cada nó *home* é responsável por gerenciar uma parcela das páginas da cache compartilhada.

B. Operações Básicas do Sistema DSMIO

O sistema DSMIO fornece 3 operações básicas: leitura, escrita e *flush*. A operação de leitura especifica a transferência de um dado da cache compartilhada para a memória

local e, no caso da primeira leitura, do disco para a *cache* compartilhada. A operação de escrita somente transfere o dado que foi modificado na memória local para a *cache* compartilhada. A operação de *flush* é executada para remover páginas da cache compartilhada por motivos de falta de espaço ou para escrever os dados da *cache* compartilhada no disco.

As operações de sincronização utilizadas para garantir acessos ao disco com exclusão mútua são idênticas às operações oferecidas por TreadMarks para garantir exclusão mútua de acesso à memória (*lock/unlock*).

B.1 Operação de Leitura

Em uma operação de leitura ao disco, o nó verifica se o dado está presente localmente, i.e., se o processador já pediu o dado ao nó *home*. Caso não esteja presente localmente, ele deve requisitar uma cópia do dado a seu respectivo nó *home*. O nó *home*, por sua vez, deve buscar o dado do disco e armazená-lo na cache compartilhada, caso o dado não esteja presente. Em seguida, ele o envia ao nó cliente.

Depois que o nó cliente recebeu o dado do *home*, ele verifica se as páginas compartilhadas nas quais o dado está contido estão inválidas, ou seja, se elas receberam invalidações por causa de *write notices*. Neste caso, os respectivos *diffs* são solicitados, aplicados ao dado e, em seguida, o dado é transferido para a área de memória local especificada pela aplicação na operação de leitura.

B.2 Operação de Escrita

Numa operação de escrita, inicialmente o nó verifica se o dado está presente localmente. Caso o dado não esteja presente, o sistema inicia uma operação de leitura DSMIO para requisitar o dado ao nó *home* e receber os seus respectivos *diffs*.

Estando o dado presente localmente, antes de transferir o dado modificado da memória local para a cache compartilhada, um *twin* é criado para a(s) página(s) em que o dado está contido. Em seguida o sistema pode então transferir o dado modificado localmente para a *cache* compartilhada e criar seus respectivos *diffs*. Após a criação dos *diffs*, a área de *twin* é liberada, assim como a área física alocada para a página compartilhada se torna disponível para ser reutilizada pelo sistema operacional e, por fim, um *write notice* é gerado para esta modificação.

O sistema DSMIO possui um mecanismo que evita os *overheads* relativos à criação de *diffs* quando os dados não são efetivamente compartilhados. Um nó *home* controla os pedidos realizados para cada dado e mantém os nós clientes informados a respeito do grau de compartilhamento dos dados. Uma operação de escrita em DSMIO, então, se adapta dinamicamente de acordo com o grau de compartilhamento do dado. Quando o dado não é efetivamente compartilhado, o sistema não cria *diffs* para ele e o envia integralmente ao nó *home* logo após sua atualização.

B.3 Operação de Flush

Há dois tipos de operações de *flush* no sistema DSMIO: *flush* parcial e *flush* total. Uma operação de *flush* parcial é realizada quando há falta de espaço na *cache* compartilhada ou na área de *diffs*. Uma operação de *flush* total é sempre executada ao término da execução da aplicação para que os dados da *cache* compartilhada sejam escritos no disco.

Quando uma operação de *flush* parcial é disparada por falta de espaço na *cache* compartilhada, o nó *home* determina a página a ser retirada, segundo um algorítmo de substituição do tipo *First in First out (FIFO)*. Caso a página a ser substituída esteja inválida, ela é recuperada e escrita de volta no disco. Caso ela seja compartilhada por outros nós do sistema, o nó *home* envia uma mensagem para que estes possam alterar os status dos respectivos dados para "não presente localmente".

Quando um *flush* parcial é disparado por falta de espaço na área de *diffs*, o processador solicita para todos os nós do sistema apenas os *diffs* correspondentes as suas páginas compartilhadas que foram gerados até o presente intervalo. Os *diffs* são aplicados nas respectivas páginas e em seguida eliminados.

Numa operação de *flush* total cada nó *home* solicita todos os *diffs* gerados para suas páginas e os aplica para recuperar a página. Estando todas as páginas recuperadas, elas são escritas de volta no disco. Atualmente, o sistema DSMIO possui apenas a operação de *flush* total. As outras operações de *flush* ainda estão em desenvolvimento.

IV. Metodologia

Para avaliar o desempenho do sistema DSMIO, utilizamos um multicomputador IBM SP com 8 nós de processamento totalmente dedicados, onde cada nó é composto de um processador Power2 com 256 Mbytes de memória e um disco local de 2Gbytes. Os nós são conectados por um *switch* de alto desempenho com banda passante nominal de 40 Mbytes/s. A instrumentação necessária para coletar os tempos medidos afeta o desempenho do sistema de forma negligível.

Como a nossa aplicação alvo é de banco de dados, estamos avaliando o nosso sistema como uma extensão de um gerente de banco de dados orientado a objetos desenvolvido pela COPPE/UFRJ, chamado Gerenciador de Objetos Armazenados, GOA [11].

Na paralelização de GOA para o ambiente *DSM*, utilizamos o sistema TreadMarks para criar um processo GOA por processador e para fornecer operações de sincronização entre processos, como *lock/unlock* e barreira. Combinando GOA e DSMIO, desenvolvemos o sistema GOA+DSMIO, onde as caches locais de cada processo são substituídas

pela cache compartilhada gerenciada por DSMIO. Para comparar GOA+DSMIO com um sistema de banco de dados distribuído tradicional, desenvolvemos o sistema GOAD, onde cada processador é servidor de uma fração da base de dados e responsável único pelos acessos à essa fração da base. Ou seja, cada vez que um processador acessa um dado da base, ele deve requisitá-lo ao seu respectivo servidor. GOA+DSMIO possui uma cache compartilhada de 4Mbytes × número de processadores (*diffs* não são armazenados na cache compartilhada; reservamos um espaco de 200Kbytes para o armazenamento do diffs na memória local). As *caches* locais de GOAD possuem 4Mbytes cada. Assim como em GOA+DSMIO, o algorítmo de substituição em GOAD é FIFO.

O desempenho de GOA+DSMIO foi avaliado através da execução de uma *traversal* do *Benchmark 007* [2]. A *traversal* utilizada foi T2 com atualizações do tipo 1 (T2/1). Paralelizamos T2 segundo o modelo SPMD de programação.

O *Benchmark 007* tem como objetivo simular aplicações do tipo CAD/CAM/CASE, não modelando em detalhe nenhuma aplicação. O objetivo do *benchmark* é testar diversos aspectos do desempenho de um gerente de banco de dados. Ele apresenta 17 aplicações, 3 das quais são aplicações de atualização a base de dados, que são variações de T2.

De uma forma simplificada, a base de dados do *Benchmark 007* é composta por objetos *AssemblyPart* que são compostos de objetos *CompositePart* que por sua vez são compostos por um conjunto de objetos *AtomicParts*. Utilizamos em nossos experimentos preliminares uma base de 100 Mbytes (base média) que é percorrida e atualizada por T2 da seguinte forma: visita-se cada objeto da base segundo a hierarquia descrita e apenas *AtomicParts* são atualizadas. É utilizada uma seção crítica diferente para cada atualização realizada em uma *CompositePart*. Os objetos compartilhados selecionados são atualizados por todos os processadores.

A aplicação T2 tem localidade temporal e espacial. Cada registro lido do disco é acessado diversas vezes pela aplicação para realizar a busca seguida da atualização dos objetos. Uma vez que estes objetos terminam de ser atualizados, o próximo registro a ser lido está localizado em um registro próximo ao que já foi lido. Neste último caso, a existência de um sistema de *prefetching* no sistema de *I/O buffer* pode alterar significativamente o desempenho da aplicação. Nossos trabalhos futuros incluem um estudo do efeito da localidade e de *prefetching* no desempenho do sistema.

Nossa avaliação preliminar de GOA+DSMIO utiliza apenas a aplicação T2. No momento, estamos estendendo essa avaliação com outras aplicações de suporte a decisão.

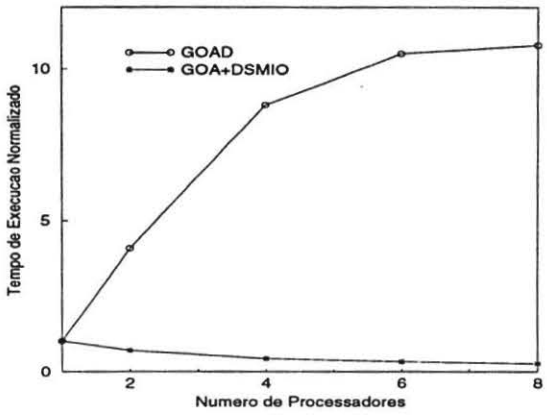

Fig. 1. Tempo de Execução T2/1 em GOAD e GOA+DSMIO

V. Análise de Resultados

Nesta seção avaliamos o desempenho do sistema GOA+DSMIO em relação ao sistema GOAD para a *traversal* T2/1. Nos concentramos apenas nos aspectos de coerência e consistência de memória dos sistemas; não incluimos aqui um estudo das possibilidades de economia de memória de DSMIO, nem do efeito da latência de escrita no disco dos sistemas, uma vez que a aplicação T2 com a base de 100Mbytes no nosso IBM-SP não nos permite isso.

Na figura 1, apresentamos as curvas do tempo de execução de T2 em GOAD e GOA+DSMIO em função do número de processadores. Os tempos estão normalizados com relação ao tempo de execução de GOAD em 1 processador.

Observamos na figura que GOA+DSMIO apresenta ganhos crescentes de desempenho em relação a GOAD à medida que aumentamos o número de processadores do sistema. Isso se deve à grande redução obtida por GOA+DSMIO no tempo de recuperação de um dado compartilhado e no tempo de propagação das modificações realizadas no dado compartilhado.

A redução obtida no tempo de recuperação do dado compartilhado se deve ao fato de que, nessa recuperação, GOA+DSMIO precisa somente receber e aplicar os *diffs* correspondentes a este dado, enquanto que GOAD necessita da última versão do dado escrita no disco. Por exemplo, o tempo de recuperação de um dado compartilhado em GOA+DSMIO é aproximadamente 5% do tempo de recuperação de um dado compartilhado em GOAD para 8 processadores.

A redução obtida no tempo de propagação das modificações realizadas no dado compartilhado é devida principalmente à diferença dos modelos de consistência de memória adotados pelos dois sistemas. GOAD emprega

o modelo *Release Consistency* e propaga as modificações realizadas no dado compartilhado para os outros processadores na saída da seção crítica. Mais especificamente, GOAD envia uma mensagem de atualização do dado para o servidor do mesmo e mensagens de invalidação para os outros processadores do sistema que compartilham o dado. O processador que escreveu no dado deve esperar *acknowledgments* de todos os processadores que receberam invalidações e um *acknowledgment* do servidor.

Já GOA+DSMIO, aproveitando as vantagens oferecidas por TreadMarks, emprega o modelo de consistência *Lazy Release Consistency* que diminui drasticamente o número de mensagens e a quantidade de dados trocados, atrasando a propagação das informações de coerência até o momento do próximo pedido de *lock*. Nesse momento, somente o processador que requisitou o *lock* recebe informações de coerência sobre os dados compartilhados. Dessa forma, GOA+DSMIO evita o excesso de tráfego gerado por mensagens de invalidação e o tempo de espera pelos *acknowledgments* e principalmente o tempo de espera pela escrita ao disco do servidor. Na verdade, a saída de uma seção crítica em GOA+DSMIO envolve apenas a criação de *diffs* e seu armazenamento em memória principal, não gerando, portanto, nenhum tipo de comunicação pela rede. Comparando o sistema GOAD e o sistema GOA+DSMIO para 8 processadores, por exemplo, vemos que o número de mensagens transmitidas no GOAD é em média 86% maior que no sistema GOA+DSMIO.

Além disso, GOA+DSMIO também reduz o tamanho das mensagens transmitidas já que atualiza os dados através dos mecanismos de *diffs*. Por exemplo, para a aplicação `T2` o tamanho das mensagens de atualização dos dados são reduzidas em média mais de 99%.

Na figura 1 observamos também que GOA+DSMIO mostrou-se escalável apenas até 4 processadores. A partir de 4 processadores, a redução no tempo de execução não é proporcional ao aumento do número de processadores, o que se deve ao uso de uma base de dados pequena em nossos experimentos. GOAD, entretanto, apresenta aumento considerável do tempo de execução com o aumento do número de processadores. Nesse caso, o problema não é o tamanho da base, mas o alto custo da coerência de dados em GOAD, que gasta aproximadamente 99% do seu tempo total de execução garantindo a coerência dos dados no disco antes de liberar cada lock, seja escrevendo os dados no disco no caso do servidor, seja enviando os dados atualizados para os seus respectivos servidores.

Nos experimentos discutidos acima existe grande compartilhamento de dados. Entretanto, numa situação em que não há compartilhamento de dados na aplicação, o uso de *diffs* para armazenamento das modificações realizadas nos dados levaria o sistema DSMIO a gerar *overhead* desnecessário de criação de *diffs* e *twins*. O que o tornaria, certamente, mais lento que GOAD. Para evitar esse problema, o sistema DSMIO possui um mecanismo que permite adaptação dinâmica em relação ao padrão de compartilhamento de dados gerado pela aplicação, conforme explicado anteriormente. De forma a avaliar essa otimização, estudamos o desempenho de GOA+DSMIO para aplicações onde não há compartilhamento de dados e observamos que GOA+DSMIO envolve *overheads* desprezíveis nesses casos.

Os resultados apresentados acima comprovam a eficiência do sistema DSMIO em trazer a coerência dos dados armazenados em disco para o nível da memória principal, mesmo inserindo no sistema *overheads* envolvidos no gerenciamento de *diffs*. Na verdade, nossos resultados mostram que esses *overheads* são suplantados pelos ganhos que esse sistema obtém através da eliminação de acessos a disco.

VI. Trabalhos Relacionados

O sistema DSMIO envolve conceitos de memória compartilhada distribuída (cacheamento cooperativo, coerência de caches e consistência de dados) na otimização de operações de E/S. Assim, nesta seção discutimos os trabalhos relacionados a cada um desses aspectos do sistema DSMIO.

Vários sistemas de memória compartilhada distribuída baseada em software já foram propostos [10, 4, 16, 14]. No entanto, DSMIO é o primeiro desses sistemas que estende o seu gerenciamento de memória principal aos dados armazenados em disco. O trabalho mais próximo ao nosso consta de uma tese recente e ainda não publicada [18]. Essa tese teve como objetivo demonstrar a facilidade de implementação do gerente de banco de dados *Postgres* sobre o sistema TreadMarks. Desta forma, o trabalho foi concentrado nas modificações necessárias a TreadMarks, sem atenção alguma ao desempenho alcançável pela combinação dos dois sistemas. Em contraste, nosso trabalho é bem mais ambicioso. Nosso foco está em investigar todas as possibilidades de melhora do desempenho da combinação dos sistemas, independente da necessidade ou não de modificações no código fonte das aplicações ou do sistema software DSM. Em termos de implementação, também existem grandes diferenças entre os trabalhos, principalmente em termos do gerenciamento da *cache* compartilhada do sistema. Nosso sistema utiliza um espaço bem menor de *cache* compartilhada, uma vez que não permite a replicação de dados em memória principal, permitindo assim uma utilização mais eficiente da memória disponível.

Assim como os sistemas de memória compartilhada distribuída, todos os sistemas de arquivos distribuídos implementam alguma forma de cacheamento de dados. As caches nesses sistemas têm sido implementadas de duas formas principais: na memória principal (como no sistema SLD [15]) ou nos discos locais (como no sistema AFS [8]). No entanto,

a principal diferença entre o nosso sistema e propostas anteriores em termos de cacheamento de dados de disco é que em DSMIO apenas as modificações (*diffs*) realizadas por um processador são cacheadas localmente e apenas uma cópia completa dos dados de disco é cacheada pelo sistema.

Em termos do modelo de consistência dos dados de disco, DSMIO também se diferencia bastante de sistemas de arquivos distribuídos anteriores. DSMIO adota o modelo *Lazy Release Consistency* [9], enquanto que a quase totalidade dos outros sistemas adota o modelo seqüencial de consistência [12, 7] e os mais recentes adotam o modelo *Release Consistency* [6, 15].

O cacheamento cooperativo [3] é implementado por vários sistemas, tais como SLD e GMS [5]. A diferença fundamental entre estes sistemas e o sistema DSMIO, está no fato de que DSMIO implementa uma cache cooperativa de *diffs*. Além disso, DSMIO só aproveita o cacheamento cooperativo em operações de leitura de dados; operações de escrita a dados são sempre realizadas localmente.

VII. Conclusões e Trabalhos Futuros

Aplicações paralelas de diversas áreas necessitam de sistemas de E/S com alto desempenho. Este artigo explora os conceitos e os mecanismos encontrados nos sistemas de memória compartilhada distribuída baseada em *software* para otimizar o *cacheamento* de dados do disco. Construímos o sistema DSMIO para um multicomputador IBM-SP e o avaliamos no contexto de um gerente de banco de dados executando aplicações de suporte e decisão.

Nossos resultados demonstram que trazer a coerência dos dados armazenados em disco para o nível da memória principal, como em DSMIO, tem ganhos significativos. Observamos reduções de até 99% no tempo de execução da traversal T2 com atualizações do *bechmark OO7*. Esses resultados demonstram que DSMIO permite suporte eficiente para aplicações em um sistema de banco de dados distribuído. Além disso, quando a aplicação não apresenta compartilhamento de dados, a adaptividade de DSMIO permite desativar o armazenamento por *diffs*, evitando a inclusão de *overheads* desnecessários.

No futuro, pretendemos realizar experimentos com bases maiores para avaliar questões de *overflow* de cache e latência de acesso ao disco (principalmente nos casos de sobrecarga do buffer de E/S do sistema operacional). Além disso, vamos avaliar a eficiência do sistema com relação à existência de múltiplos escritores concorrentes aos dados do disco, economizando espaço de armazenamento em cache e eliminando sincronizações.

Necessitamos também fornecer confiabilidade ao gerente de banco de dados, seja através da criação e gerenciamento de discos de *log* das operações realizadas, seja implementando DSMIO com um *software DSM* tolerante a falhas.

References

[1] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless Network File Systems. In *Proc of the 15th Symposium on Operating System Principles*, 1995.

[2] M. Carey, D. DeWitt, and J. Naughton. The 007 Benchmark. In *Proc of the ACM SIGMOD International Conference on Management of Data*, 1993.

[3] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *Proc of the 1st Symposium on Operating Systems Design and Implementation*, 1994.

[4] L. R. Monnerat and R. Bianchini. Efficiently Adapting to Sharing Patterns in Software DSMs. In *Proc of the 4th IEEE Symposium on High-Performance Computer Architecture*, 1998.

[5] M. Feely, W. Morgan, F. Poghin, A. Karlin, H. Levy, and C. Thekath. Implementing Global Memory Management in a Workstation Cluster. In *Proc of the 15th Symposium on Operationg Systems Principles*, 1995.

[6] G. Gibson, D. Stodolsky, F. Chang, W. CourtrightII, C. Demetriu, E. Ginting, M. Holland, Q. Ma, L. Neal, R. Patterson, J.Su, R.Youssef, and J.Zelenka. The Scotch Parallel Storage Systems. In *Proc of the IEEE CompCon conference*, 1995.

[7] J. Hartman and J. Outerhout. The Zebra Striped Network File System. In *Proc od the 14th Symposium on Operating Systems Principles*, 1993.

[8] J. Howard, M. Kazar, S. Meness, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM transactions on Computer Systems*, 6, 1988.

[9] P. Keleher, A.L. Cox, and W. Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In *Proc of the 19th Annual International Symposium on Computer Architecture*, 1992.

[10] P. Keleher, S. Dwarkadas, A.L. Cox, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proc of the 1994 Winter Usenix Conference*, 1994.

[11] L. Meyer and M. Mattoso. Paralelismo em SGBDOO com Memória Distribuída: Uma Análise do Desempenho do ParGOA-MD. In *Anais do XII Simpósio Brasileiro de Banco de Dados*, 1997.

[12] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch. The Sprite Network Operating System. *IEEE Computer*, 2, 1988.

[13] A. Purrakayastha, C. Ellis, and D. Kotz. ENWRICH: A Compute-Processor Write Caching Scheme for Parallel File Systems. In *Proc of the 4th Workshop on I/O in Parallel and Distributed Systems*, 1996.

[14] C. B. Seidel, R. Bianchini, and C. L. Amorim. The Affinity Entry Consistency Protocol. In *Proc of the 1997 International Conference on Parallel Processing*, 1997.

[15] R. Shillner and E. W.Felten. Simplifying Distributed File Systems Using Shared Logical Disk. In *Proc. of the Intel Supercomputer User's Group Conference*, 1995.

[16] Y. Zhou, L. Iftode, and K. Li. Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Memory Virtual Memory Systems. In *Proc of the 2nd Symposium on Operating Systems Design and Implementation*, 1996.

[17] A. Purrakayastha, C. Ellis, D. Kotz,N. Nieuwejaar, and M. Best. Characterizing parallel file-access patterns on a large-scale multiprocessor. In *Proceedings of the International Parallel Processing Symposium* 1995.

[18] T. Parker I/O-Oriented Applications on a Software Distributed-Shared Memory System. Master of Science Thesis - Computer Science Dept-Rice university, 1999.