Técnicas para Categorização de Padrões de Compartilhamento em Sistemas *Software DSM*

M. C. S. de Castro, [1] C. L. de Amorim [2*]

[1] Depto de Ciência da Computação/UFJF
[2] COPPE - Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511 - CEP 21945-970 - Rio de Janeiro - RJ
clicia,amorim@cos.ufrj.br

Abstract—

In this paper, we introduce a new categorization algorithm called RITMO to determime precisely sharing patterns in software distributed shared memory (DSM). RITMO is based on the page sharing state information generated at run-time by a finite state machine (FIESTA). RITMO categorizes dynamicaly pages according to their inherent sharing patterns in contrast with the induced sharing patterns used in current algorithms. The main advantage of this approach is to enable RITMO to categorize accurately irregular applications in which fine-grain memory accesses often mask inherent sharing patterns making categorizations based on induced patterns inexact. Both FIESTA and RITMO are transparent to the programmer and do not require compiler or hardware assistance. Furthermore, they add no extra messages to the protocol and intrusion is kept extremely low.

Through detailed execution-driven simulation of six benchmark applications running on TreadMarks with 8 processors we show that RITMO categorization is highly precise on classifying pages that are responsible for more than 93% of the application page faults, as either single-writer or migratory in five out of six benchmarks. In both irregular applications tested, 50% (in Water-Nsq) and 8% (in Barnes2) of pages which correspond to 26% and 60% of the application page faults respectively, have induced sharing patterns different from their inherent sharing patterns which were precisely determined by RITMO. Also, our results show that prefetching and forwarding could be efficiently implemented due to the RITMO accuracy, reducing remote data overheads from 14% (in BarnesTmk) up to 90% (in Em3d). As a result, the application execution time improved from 7% (Barnes2) to 38% (Em3d). Our conclusion is that RITMO based on FIESTA is a effective technique to improve software DSM performance in a large class of applications.

Keywords— **distributed shared memory systems, sharing patterns, latency**

I. Introdução

Nos últimos anos, observa-se a crescente expansão de aplicações científicas e comerciais que se beneficiam do grande poder computacional de sistemas de computação paralela. Em especial, os sistemas *software DSM* (*Distributed Shared Memory*) têm recebido grande atenção da comunidade científica porque combinam a facilidade do modelo de programação de memória compartilhada com o baixo custo de sistemas de memória distribuída. Entretanto, tais sistemas podem ter seu desempenho limitado devido à latência de comunicação e as operações realizadas para manter a memória coerente.

Diferentes técnicas de tolerância à latência vêm sendo implementadas para se reduzir os *overheads* de protocolos *software DSM*. Isto porque reduzem o impacto causado pelo falso compartilhamento[1] e pela fragmentação[2], que são causados pelo grande tamanho da página, que tipicamente é utilizada como unidade de coerência em sistemas *software DSM*.

As técnicas de tolerância à latência, tais como *prefetching* e *forwarding*, visam reduzir o tempo de espera de um processador por dados remotos, buscando ou enviando antecipadamente os dados compartilhados. Estas técnicas implementam mecanismos de previsão de falhas e são baseadas em anotações inseridas pelo programador/compilador ou em ações do *run-time system* ou combinação destas. Em geral, elas se baseiam em métodos que categorizam o padrão de compartilhamento das páginas, de forma aproximada, para realizar as previsões. Aumentando-se a precisão desses métodos poderemos incrementar a eficiência das técnicas, e portanto, reduzir os *overheads* de protocolos *software DSM*. A precisão da categorização do padrão de compartilhamento é fundamental para que o protocolo execute ações que reduzam ao máximo a quantidade de mensagens trocadas para manter a coerência dos dados compartilhados e minimizar o tempo de espera pelos dados.

Neste artigo introduzimos o algoritmo RITMO para categorizar padrões de compartilhamento em sistemas *software DSMs* de forma precisa e detalhada. Este algoritmo foi desenvolvido baseado nas informações fornecidas por FIESTA, uma nova máquina de estados finitos baseada em eventos de coerência registrados em tempo de execução[5]. Ambos, FIESTA e RITMO, oferecem as vantagens de não aumentar o

*Trabalho parcialmente financiado pela Finep, CNPq e CAPES

[1] O falso compartilhamento ocorre quando dois ou mais processadores realizam acesso a dados não relacionados que estão armazenados numa mesma página, e pelo menos um deles realiza uma escrita.

[2] A fragmentação ocorre quando um processador necessita somente de parte da página mas tem que carregá-la inteira, causando tráfego de dados desnecessário.

número de mensagens do protocolo DSM, não serem intrusivos à aplicação e serem transparentes ao programador.

RITMO categoriza eficientemente os padrões de compartilhamento tanto em aplicações regulares como em aplicações irregulares. Mais especificamente, mostramos que RITMO é capaz de detectar padrões mais precisos em páginas com acessos de escrita irregulares: (1) quando ocorrem dentro e fora de seções críticas, e (2) quando ocorrem em seções críticas guardadas por *locks* distintos. Essas páginas são consideradas simplificadamente como tendo múltiplos escritores pelas atuais estratégias de categorização do padrão de escrita[1, 8], restringindo a adaptabilidade do protocolo e a eficiência das técnicas de tolerância à latência. Mostramos que observando a seqüência de estados de compartilhamento dessas páginas, podemos categorizá-las com padrões único escritor, múltiplos escritores ou migratórias, retirando essa limitação.

Nossos resultados experimentais são baseados na implementação de FIESTA, de RITMO e de estratégias de tolerância à latência, utilizando um simulador de TreadMarks[7] num multicomputador com oito processadores. Nossos resultados mostram que RITMO não aumenta o número de mensagens transferidas e o aumento da quantidade de *bytes* transmitidos para a maioria das aplicações é insignificante. Além disso, as técnicas de tolerância à latência produziram uma redução significativa nos *overheads* de acesso a dados remotos do protocolo, reduzindo entre 7% e 38% o tempo total de execução.

Este artigo esta organizado da seguinte forma. A próxima seção descreve o sistema *software DSM* TreadMarks no qual baseamos nossos experimentos. A seção III introduz o algoritmo de categorização RITMO. As técnicas de tolerância à latência, o ambiente de simulação e as aplicações estão descritos na seção IV. A seção V analisa o comportamento de cada aplicação segundo as informações fornecidas por RITMO. A seção VI discute os resultados de desempenho obtidos. A seção VII resume os trabalhos relacionados e apresentamos nossas conclusões na seção VIII.

II. Sistema TreadMarks

TreadMarks (TM) é um sistema de memória compartilhada distribuída implementado em *software*. A unidade de coerência é a página, sendo a coerência dos dados mantida com uso do protocolo de invalidações. Este protocolo é implementado através da propagação de *write notices* em operações de sincronização.

Para reduzir os efeitos de falso compartilhamento, TM fornece suporte a múltiplos escritores através dos mecanismos de *twining* e *diffing*. Inicialmente todas as páginas são protegidas contra escrita. Quando há uma tentativa de atualização numa página, é gerada uma falha de acesso. O TM intercepta esta falha, faz uma cópia da página (*twin*) e a libera para escrita. Quando for necessária a propagação das modificações, TM faz uma comparação entre o *twin* gerado e a versão modificada, e cria um *diff* contendo todas as modificações locais feitas. A cada *diff* existe um *write notice* associado identificando o intervalo e o processador onde o *diff* deve ser criado.

Com o objetivo de reduzir o tráfego de dados pela rede, TM adota o modelo de consistência *lazy release consistency*. A execução de uma aplicação é dividida em intervalos que são iniciados nas sincronizações entre os processsadores. O envio e a aplicação dos *diffs* são atrasados (*lazy*) até os próximos pontos de sincronização.

Nas operações de sincronização as páginas que foram modificadas por outros processadores são invalidadas. Quando ocorre uma falha num acesso à página é necessário buscar um conjunto de *diffs* para torná-la válida. Mais detalhes podem ser obtidos em [7].

III. RITMO

Para a captação dos estados de compartilhamento, FIESTA não altera o modelo de consistência LRC e nem o protocolo de coerência. Os eventos utilizados para provocar as transições de estado são os mesmos eventos utilizados pelo protocolo para manter a memória coerente. Além disso, ela não inclui mensagens extras para a captação dos estados de compartilhamento, o que é fundamental para sistemas *software DSM*. A máquina de estados completa pode ser encontrada em[5].

Para destacar mais facilmente a diferença entre outras abordagens e a nossa, mostramos na figura 1 cinco grupos diferentes de estados considerando somente os acessos de escrita: UCLP, SW, MW, MIGR e INOUT. O grupo SW é composto pelas páginas onde ocorrem acessos de escrita realizados por um único processador. As páginas onde ocorrem acessos de escrita por mais de um processador pertencem ao grupo MW. As páginas onde ocorrem acessos de escrita realizados por mais de um processador mas de modo exclusivo, estão no grupo MIGR. Este grupo também inclui as páginas que têm acessos de escrita exclusivos com mais de uma variável de *lock*. As páginas pertencentes ao grupo INOUT são aquelas onde ocorrem tanto acessos dentro quanto fora de seção crítica. O grupo UCLP representa o estado inicial das páginas.

Na figura 1 destacamos com um retângulo a contribuição de FIESTA. As máquinas de estados atuais não conseguem distingüir e categorizar de forma precisa as páginas com padrões de escrita INOUT ou MIGR com mais de uma variável de *lock*. A nossa solução é capturar a seqüência dos estados de compartilhamento através de FIESTA e aplicar RITMO na seqüência para categorizar o padrão de escrita da página em três grupos distintos: SW, MW e MIGR. Nestas categorias podem ou não existir leitores.

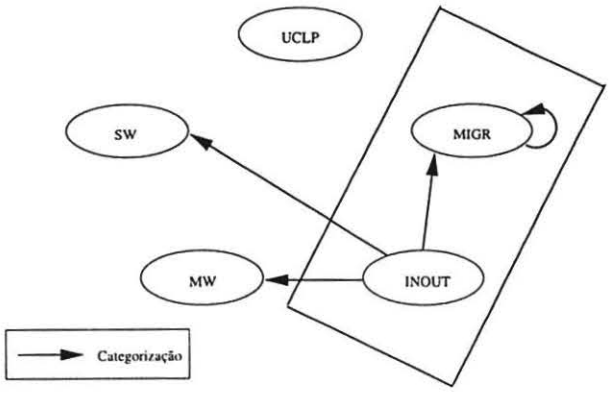

Fig. 1. Estados de compartilhamento agrupados, baseados no modelo de FIESTA

Categorização MIGR - Página 9

Acessos à lock		{R}	{W}	{P}	{C}
1	INOUT	{2, 5, 6, 7}	{1}	{1}	{1}
2	INOUT	{2, 5, 6, 7}	{1}	{1}	{1}
3	INOUT	{2, 5, 6, 7}	{1}	{1}	{1}
4	INOUT	{2, 5, 6, 7}	{1}	{1}	{1}
5	INOUT	{2, 5, 6, 7}	{1}	{1}	{1}
6	INOUT	{2, 5, 6, 7}	{1}	{1}	{1}
7	INOUT	{5, 6, 7}	{1}	{1, 2}	{2}
8	INOUT	{5, 6, 7}	{1}	{1, 2}	{2}

Fig. 2. Exemplo de seqüência de estados de compartilhamento para uma página INOUT categorizada como MIGR

Para páginas com predominância de acessos fora de seção crítica, gravamos a seqüência de estados globais na barreira até encontrarmos os três primeiros estados globais onde ocorreram escritas. Após atingir esta condição, analisamos a seqüência dos estados de compartilhamento e a categorizamos.

Para páginas com predominância de acessos dentro de seção crítica, gravamos a seqüência de estados parciais na liberação das variáveis de *lock* durante os primeiros oito acessos à página. Da mesma forma que para acessos fora de seção crítica, após atingir essa condição, analisamos a seqüência dos estados de compartilhamento e categorizamos as páginas. Para páginas onde existe tanto acessos dentro quanto fora de seção crítica, analisamos e categorizamos o padrão de compartilhamento de acordo com a primeira condição atendida: três estados globais onde ocorreram escritas ou oito acessos dentro de seção crítica. Os valores de três estados globais ou oito acessos dentro de seção crítica são independentes das aplicações, e foram escolhidos após observarmos o comportamento de 18 aplicações diferentes[3].

Exemplificamos na figura 2 uma seqüência de estados de compartilhamento para uma página com acesso irregular e que gera categorização MIGR. Uma seqüência de eventos semelhante a apresentada ocorre na aplicação Water-Nsquared e reflete a visão que o processador que está liberando o *lock* tem da página.

O que caracteriza o estado de compartilhamento são os tipos de acesso realizados na página e o conjunto de processadores que realizam os acessos. Essas informações, são armazenadas nos conjuntos R, W, P e C, que representam acessos de leitura e escrita dentro (R e W) e fora de seção crítica (P e C), respectivamente.

Desde o primeiro estado gravado, a página já está no estado INOUT. O processador P_1, que conhece que outros processadores fizeram acessos fora de seção crítica, escreve na página dentro de seção crítica. Este mesmo processador escreve na página sob variáveis de *lock* diferentes até o sexto acesso. Nos dois estados seguintes, é o processador P_2 que passa a escrever na página. Como o único processador dentro do conjunto W também pertence ao conjunto P, esta página é categorizada como MIGR.

IV. Metodologia Experimental

A. Técnicas de Tolerância à Latência

Utilizamos neste estudo as técnicas de *forwarding* (FWD) e de *prefetching* (PRF), no protocolo TreadMarks, para avaliar o potencial de RITMO. A menos que explicitamente citado, nas sincronizações de barreira ou com *locks* são executadas as mesmas operações. A diferença está no tratamento das falhas de acesso.

A técnica FWD é utilizada para tolerar os *overheads* relativos aos mecanismos de *twining* e *diffing*. Ela habilita o protocolo a enviar ou não páginas antecipadamente. Na nossa implementação, as páginas são enviadas antecipadamente pelo escritor a um conjunto de processadores de modo seletivo, isto é, a um conjunto de processadores com os quais sincroniza, que têm as páginas invalidadas e que se prevê que sofrerão falha de acesso às páginas. Ela pode ser ativada tanto nas sincronizações de barreira quanto nas sincronizações por *locks* e as páginas são enviadas antecipadamente somente se sua categorização for SW ou MIGR. As páginas MW são tratadas segundo os mecanismos de *twinning* e *diffing*.

O envio antecipado de uma página para outros processadores é realizado utilizando um conjunto de previsão. O

[3]Essas aplicações pertencem aos *benchmarks* SPLASH-2 (Barnes, FMM, Ocean, Radiosity, Raytrace, Water Nsquared, Water Spatial, Volrend, Cholesky, FFT, LU, Radix), NAS (Appbt, IS), pacote de distribuição do Treadmarks (BarnesTmk eTSP), além das aplicações Em3d, desenvolvida na Universidade de Berkeley, e SOR, desenvolvida na COPPE/Sistemas - UFRJ. Para definirmos como condição três estados globais ou oito acessos dentro de seção crítica comparamos a categorização assumida por RITMO para cada página com o padrão de compartilhamento exibido pelas páginas ao longo de toda a execução das aplicações

conjunto é obtido baseando-se no estado global da página e no conjunto de processadores das seqüências de estados utilizadas para a sua categorização.

Na saída da barreira, cada processador, para cada página que compartilha, verifica sua categoria. Se a página for SW ou MIGR e o processador é o único escritor, então, se existem leitores no conjunto de previsão com a página inválida, esta é enviada antecipadamente. Da mesma forma, num pedido de aquisição de *lock*, o processador *owner* verifica a categoria das páginas ao enviar a posse do *lock* ao processador *acquirer*. As páginas SW ou MIGR que ele é escritor, são enviadas antecipadamente se as condições forem atendidas.

A técnica PRF visa minimizar o *overhead* do acesso a dados remotos, antecipando a busca destes dados. Os dados invalidados por outros processadores, são trazidos antecipadamente para que no momento do acesso à página, aumente a chance dos *diffs* já estarem disponíveis. Neste trabalho, a técnica de *prefetching* é implementada somente na barreira. Nenhuma ação é realizada para falhas que ocorram dentro de seções críticas. Ela também utiliza o conjunto de previsão para requisitar os dados de modo seletivo. A requisição só é disparada para páginas categorizadas como MW.

Os dados recebidos antecipadamente são armazenados e quando ocorre uma falha de acesso, é verificado se todos os *diffs* estão disponíveis. Se necessário, são coletados *diffs* adicionais, que não tenham sido solicitados, ou que não tenham sido ainda recebidos. Todos os *diffs* são aplicados cronologicamente à página, de modo a torná-la válida.

B. Ambiente de Simulação

Simulamos uma rede de estações de trabalho com 8 nós utilizando Mint[9]. Cada nó consiste de um processador, um *write buffer*, uma *cache* de dados de primeiro nível mapeada diretamente, um módulo de memória local, e um roteador de rede em malha, utilizando o roteamento *wormhole*. Consideramos que todas as instruções são executadas em um ciclo.

A tabela I mostra os parâmetros utilizados em nossas simulações, correspondendo a valores compatíveis com processadores e redes de interconexão atuais.

C. Aplicações

Para demonstrar o potencial de RITMO juntamente com as técnicas de tolerância à latência utilizamos seis aplicações com características distintas. As aplicações Barnes2, FFT e Water-Nsquared pertencem ao SPLASH-2 *suite*[10]. BarnesTmk faz parte do pacote de distribuição de TreadMarks. A aplicação Em3d foi desenvolvida na Universidade de Berkeley[4] e SOR foi desenvolvida localmente. Nas referências citadas podem ser encontrados mais detalhes sobre esses programas.

TABELA I
PARÂMETROS DO SISTEMA SIMULADO. 1 CICLO = 5 NS.
* = CICLOS/PALAVRA

Constantes do Sistema	Valores
Número de processadores	8
Tamanho da TLB	128 entradas
Tempo para completar a TLB	100 ciclos
Todas as interrupções	4000 ciclos
Tamanho da página	4K bytes
Cache total por processador	256K bytes
Tamanho do *buffer* de escrita	4 entradas
Tamanho da linha da *cache*	32 bytes
Ativação da memória	18 ciclos
Acesso à memória	1.25*
Ativação do PCI	18 ciclos
Acesso PCI *burst*	3*
Largura da rede	16bits (bidirecional)
Overhead de mensagens	400 ciclos
Latência de *switch*	4 ciclos
Latência de *wire*	2 ciclos
Processamento de listas	6 ciclos por elemento
Geração de *twin*	5* + acessos à memória
Criação e aplicação de *diff*	7* + acessos à memória

V. ANÁLISE DO COMPORTAMENTO DAS APLICAÇÕES

A. Overheads de FIESTA e RITMO

A tabela II mostra o aumento na quantidade de *bytes* transmitidos e o aumento no tempo de execução de cada aplicação quando inserimos o algoritmo de categorização, em relação à versão original de TreadMarks. É importante observar que nem FIESTA e nem RITMO aumentam o número de mensagens transmitidas.

TABELA II
QUANTIDADE DE *BYTES* TRANSMITIDOS E TEMPO DE EXECUÇÃO

Aplicação	*Bytes*	Tempo
BarnesTmk	1%	1%
Em3d	10%	1,5%
FFT	1%	1%
SOR	10%	0,5%
Water-Nsq.	1%	1%
Barnes2	1%	1%

Como podemos observar nessa tabela, somente para as aplicações Em3d e SOR há um aumento significativo para a quantidade de *bytes* transmitidos devido a grande quantidade de páginas compartilhadas e/ou de barreiras. Nas barreiras é computado o estado global das páginas. Apesar do aumento de *bytes* transmitidos, o impacto no tempo de execução é bem

pequeno e não atinge 1,5% em Em3d e é inferior a 0,5% em SOR.

B. Categorização dos Padrões de Escrita

Na tabela III destacamos os padrões de escrita das aplicações e quantificamos a influência de cada padrão no desempenho da aplicação, mostrando a porcentagem de falhas de página (**Falha**) relativo a cada grupo de páginas (**Pag**) categorizado por RITMO.

TABELA III
PORCENTAGEM DE PÁGINAS E DE FALHAS DE ACESSO POR CATEGORIA

Aplicação	SW		MIGR		MW	
	Pag	Falha	Pag	Falha	Pag	Falha
BarnesTmk	27	19	-	-	73	81
Em3d	11	93	-	-	89	7
FFT	67	95	-	-	33	5
SOR	99,7	99,6	-	-	0,3	0,4
Water-Nsq.	14	0,5	85	94,4	0,5	0,1
Barnes2	5	60	93	35	2	5

Baseados nesses resultados, a escolha das técnicas de tolerância à latência mais apropriadas é naturalmente identificada. A aplicação BarnesTmk é a única aplicação com quantidade de falhas de página significativa em duas categorias diferentes, SW e MW, sugerindo que avaliemos o potencial de ambas as técnicas *forwarding* e *prefetching*. As aplicações Em3d, FFT e SOR têm predominância de falhas de página na categoria SW, já as aplicações Water-Nsquared e Barnes2 têm a predominância da categoria MIGR, o que sugere que avaliemos o potencial da técnica de *forwarding*, para reduzir o tempo de espera pelos dados.

C. Padrões de Escrita Irregulares

A tabela IV mostra a porcentagem de páginas com padrão de escrita irregular e de falhas de acesso geradas por estas páginas, para as aplicações Water-Nsquared e Barnes2. Nas demais aplicações do nosso conjunto de testes não foram encontrados padrões de escrita irregulares.

TABELA IV
PORCENTAGEM DE PÁGINAS E DE FALHAS DE ACESSO EM PÁGINAS COM PADRÃO IRREGULAR

Aplicação	INOUT/MULT-LOCKS	Falhas
Water-Nsq.	50	26
Barnes2	8	60

Como podemos observar, para ambas as aplicações, a quantidade de falhas das páginas irregulares é substancial, destacando a importância de RITMO baseado nas informações de FIESTA.

VI. Análise de Desempenho das Aplicações

Neste trabalho estamos interessados em avaliar a redução dos *overheads* causados pelo acesso a dados remotos quando introduzimos as estratégias FWD e PRF, baseado nas categorizações de RITMO. Para isto, mostramos na figura 3 o perfil de execução das aplicações. As barras da esquerda e da direita de cada grupo correspondem, respectivamente, à versão original de TM e a nossa melhor versão (RITMO + FWD ou FWD+PRF).

Cada barra mostra o tempo de execução dividido em: *busy* (trabalho útil); *data* (tempo de processamento de operações de coerência e da latência da rede envolvidos na busca de páginas ou *diffs*); *synch* (tempos de barreiras e de *locks*); ipc (tempo que o processador gasta servindo pedidos remotos) e *others* (latências de falhas na *cache* e de falhas na TLB e tempos de espera para o *write buffer* esvaziar e de interrupção).

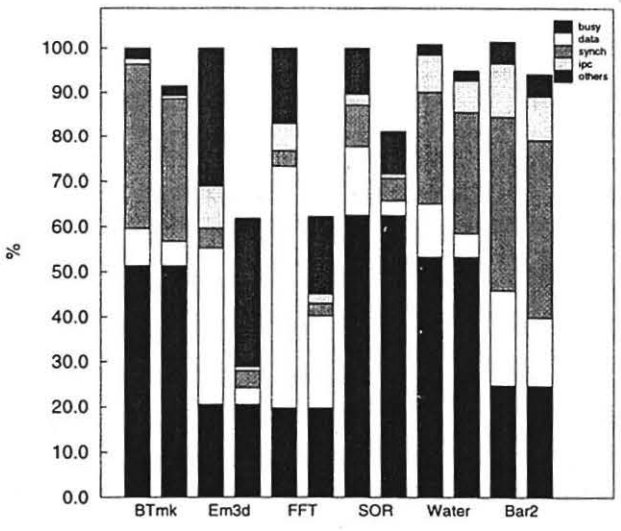

Fig. 3. Perfil de Execução das Aplicações

Observando a figura 3 vemos que muitas das aplicações sofrem severamente com os *overheads* de acesso a dados remotos e de sincronização. O percentual de tempo de espera pelos dados remotos é significativo em Em3d (35%), FFT (53%) e Barnes2 (20%), enquanto que o percentual de *overhead* de sincronização é significativo em BarnesTmk (36%), Water-Nsquared (25%) e Barnes2 (38%). Assim, o potencial de eliminação direta de *overheads* (valores de *data*) varia entre 8% (BarnesTmk) e 53% (FFT).

Verifica-se que para todas as aplicações, reduzimos o tempo de acesso a dados remotos, entre 14% em BarnesTmk e 90% em Em3d, sendo que para as outras aplicações a

redução ficou acima de 30%. A redução obtida em Em3d se deve a alta taxa de acerto do padrão de categorização inferido e do conjunto de processadores que compartilham as páginas. Em BarnesTmk a redução só não foi maior porque nas nossas estratégias de implementação das técnicas FWD e PRF, na saída da barreira, enviamos as páginas ou requisitamos os *diffs* antecipadamente. Um dos efeitos da combinação destas técnicas é aumentar a quantidade de tráfego na rede na saída da barreira, o que anula um pouco o ganho das técnicas. Reduzimos, também, para todas as aplicações o tempo total de execução das aplicações entre 7% para Barnes2 e 38% para Em3d.

VII. Trabalhos relacionados

Protocolos adaptativos podem implementar diferentes modelos de consistência, adaptação tais como atualização ou invalidação e adaptação entre um e vários escritores[2, 6, 1, 8, 3].

Existem dois trabalhos mais próximos ao nosso.Em[1] são identificados padrões de escrita através de mensagens de posse e de *write notices* com *version number*. As diferenças principais são: não usamos mensagens extras para a identificação dos padrões de escrita e páginas com acessos a seções críticas realizados com *locks* diferentes são consideradas MW, enquanto RITMO pode categorizá-las como SW, MW ou MIGR. Em[8], a diferença entre RITMO e o algoritmo de categorização SPC é que categorizamos páginas onde ocorrem seqüências de estados de compartilhamento com acessos dentro e fora de seção crítica ou com acessos a seções críticas realizados com *locks* diferentes, como SW, MW ou MIGR, enquanto SPC as mantém como MW.

VIII. Conclusões

Neste artigo introduzimos RITMO, um novo algoritmo de categorização que identifica eficazmente os padrões de escrita tanto regulares quanto irregulares, utilizando uma máquina de estados finitos para *software DSM*. Essa categorização é realizada em tempo de execução e de modo transparente ao usuário. Avaliamos o potencial de RITMO aplicando-o ao protocolo TreadMarks para um sistema com 8 processadores, e utilizando técnicas de tolerância a latência FWD e PRF, baseadas em RITMO.

Nossos resultados mostraram que RITMO tem *overhead* insignificante para as aplicações utilizadas, independente de apresentarem padrões de escrita regulares ou irregulares. Além disso, nossos resultados demonstraram que, baseados nas informações de RITMO, as técnicas implementadas com mecanismos de previsão, FWD e PRF proporcionam a redução substancial dos *overheads* de espera pelos dados remotos e seus efeitos indiretos. A nossa conclusão é que a utilização de RITMO conjugado com técnicas de tolerância à latência, é uma opção que deve ser considerada na implementação de protocolos *software DSM*.

Referências

[1] C. Amza, A. Cox, S. Dwarkadas, and W. Zwaenepoel. Software DSM Protocols that Adapt Between Single Writer and Multiple Writer. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA-3)*, pages 261 – 271, Febrary 1997.

[2] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Implementation and Performance of Munin. In *Proc. of the 13th ACM Symp. on Operating Systems Principles*, pages 152–164, October 1991.

[3] M. C. S. Castro and C. L. Amorim. Avaliação do Potencial de Técnicas Adaptativas Conjugadas para Software DSMs. In *Anais do Simpósio Brasileiro de Arquitetura de Computadores - PAD*, pages 9 – 19, September 1998.

[4] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programing in Split-C. In *Proc. of Supercomputing'93 (SC'93)*, pages 262 – 273, November 1993.

[5] Maria Clicia Stelling de Castro. *Técnicas para Detecção e Exploração de Padrões de Compartilhamento em Sistemas de Memória Compartilhada Distribuída*. PhD thesis, Programa de Engenharia de Sistemas e Computação - COOPE/UFRJ, Dezembro 1998.

[6] S. Dwarkadas, P. Keleher, A. L. Cox, and W. Zwaenepoel. Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology. In *Proc. of the 20th An. Int'l Symp. on Computer Architecture (ISCA'93)*, May 1993.

[7] 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*, January 1994.

[8] L. R. Monnerat and R. Bianchini. Efficiently Adapting to Sharing Patterns in Software DSMs. In *Proc. of the 4th IEEE Symp. on High-Performance Computer Architecture (HPCA-4)*, pages 289 – 299, Febrary 1998.

[9] J. E. Veenstra and R. J. Fowler. MINT: A Front end for Efficient Simulation of Shared-Memory Multiprocessors. In *Proc. of the 2nd Int'l Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1994.

[10] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH2 Programs: Characterization and Methodological Considerations. In *Proc. of the 22nd An. Int'l Symp. on Computer Architecture (ISCA'95)*, pages 24–36, May 1995.