Algoritmos de Consistência para Softwares DSM Baseados em Objetos

Christiane V. Pousa[1], Luís F. W. Góes[1], Luiz E. S. Ramos[2], Carlos A. P. S. Martins[1]
[1]*Pontifícia Universidade Católica de Minas Gerais – Programa de Pós Graduação de Engenharia Elétrica – Grupo de Sistemas Digitais e Computacionais*
Belo Horizonte – Minas Gerais - Brasil
[2]*Rutgers University – New Jersey - New Jersey – Estados Unidos*
pousa@ieee.org, lfwgoes@pucminas.br, luizesramos@ieee.org,capsm@pucminas.br

Resumo

Softwares de memória compartilhada distribuída (DSM – Distributed Shared Memory) permitem que os nós de uma arquitetura paralela distribuída compartilhem dados. Este compartilhamento permite que os nós tenham acesso concorrente/paralelo a um mesmo conjunto de dados. Então, torna-se necessário que os nós saibam quando e como realizar esses acessos sem gerar inconsistência nos dados compartilhados. Um algoritmo de consistência é responsável por garantir que nenhum nó da arquitetura paralela distribuída tenha acesso a um dado compartilhado inválido ou bloqueado. Neste artigo, analisamos o desempenho de cinco algoritmos de consistência uniforme para softwares DSM baseados em objetos, através de simulação, em um computador paralelo. Para análise de desempenho nós usamos aplicações sintéticas e aplicações reais. Os resultados mostram que os algoritmos de consistência com mecanismos de reconfiguração possuem melhor desempenho. Além disso, nós verificamos que para a maioria das aplicações simuladas o protocolo de coerência é o principal responsável pelo desempenho.

1. Introdução

Nos últimos anos, com o aparecimento das linguagens de programação orientadas por objetos, novos sistemas de memória compartilhada distribuída (DSM) têm sido desenvolvidos por meio de *software*. Estes novos *softwares* DSM são baseados, em sua grande maioria, em objetos e permitem que aplicações paralelas sejam desenvolvidas usando o conceito de objeto compartilhado. Nestes sistemas, a sobrecarga de comunicação pode ser muito alta, se os objetos compartilhados são movimentados entre os nós do aglomerado de computadores com grande freqüência. A replicação dos objetos entre os nós é uma das técnicas usadas para reduzir a sobrecarga de comunicação e assim, alcançar maior desempenho. Entretanto, a replicação de objetos gera problemas (comunicação, armazenamento dos dados, etc) e o maior deles é a manutenção da consistência dos objetos compartilhados entre os nós [1] [2].

Um algoritmo de consistência é responsável por garantir que nenhum nó da arquitetura paralela tenha acesso a um objeto compartilhado inválido ou bloqueado, com a menor sobrecarga de comunicação possível. Ele determina as regras de acesso, ordem e manipulação dos objetos compartilhados representando, assim, um modelo de consistência de memória. Então, o algoritmo de consistência consegue manter os estados das réplicas dos objetos compartilhados consistentes para todas as aplicações [3] [4] [5]. Alguns algoritmos de consistência foram propostos na literatura [1] [2] [3] [4] [5] [6] [7] [8] [9] [10], e apresentados como soluções eficientes para o problema de consistência em *softwares* DSM.

Muitos trabalhos e estudos de análise de desempenho de algoritmos de consistência têm sido realizados [11] [12] [13] [14] [15] [16] [17] [18]. Entretanto, a maioria desses utilizam modelos analíticos e não consideram algoritmos de consistência, com e sem mecanismos de reconfiguração, para *softwares* DSM baseados em objetos.

Neste artigo, nós analisamos o desempenho de cinco algoritmos de consistência uniforme para *softwares* DSM baseados em objetos [1] [2] [5] [7] [8] [9] [10] mais difundidos, usando simulação, em um computador paralelo usando aplicações sintéticas e reais.

O artigo está organizado da seguinte forma: a seção 2 apresenta os cinco algoritmos de consistência selecionados. A seção 3 discute os principais trabalhos relacionados. A seção 4 apresenta o método experimental utilizado para a análise de desempenho. A seção 5 apresenta os resultados obtidos e a seção 6 apresenta as conclusões e discute possíveis trabalhos futuros.

2. Algoritmos de Consistência

Como apresentamos na introdução, os algoritmos de consistência devem combinar uma semântica simples e uma sobrecarga de comunicação baixa. Por esse motivo, dentre os trabalhos encontrados, nós selecionamos [7] [8] [5] [9] [10] [1] [2] para fazer a análise de desempenho. Estes trabalhos foram selecionados porque propõem e apresentam algoritmos de consistência que implementam modelos de consistência com semântica simples (sem primitivas de sincronização como os modelos uniformes) utilizando técnicas que visam diminuir a sobrecarga de comunicação em *softwares* DSM baseados em objetos.

Nos trabalhos [7] [8], Raynal propõe um novo algoritmo para implementar o modelo de consistência seqüencial. Nestes trabalhos, o autor mostra que o modelo de consistência seqüencial pode ser visto, no nível de implementação, como um modelo de consistência atômica preguiçosa. O algoritmo de Raynal propõe uma nova restrição de consistência, OO (operações conflitantes no mesmo objeto), que não permite que operações conflitantes (leitura-escrita e escrita-escrita) em um mesmo objeto sejam executadas em paralelo. Com esse tipo de restrição o número de operações sequencializadas diminui em relação aos outros tipos de restrições, e conseqüentemente o número de processos bloqueados durante a execução das aplicações torna-se menor. O algoritmo também permite replicação total dos objetos compartilhados nos nós do aglomerado e usa protocolo de coerência de invalidação na escrita para manter os objetos coerentes entre os nós.

Em [5], Zhou, apresenta um *software* DSM baseado no modelo de consistência de memória seqüencial. O algoritmo de consistência proposto possui menor sobrecarga de comunicação que outros algoritmos propostos na literatura, porque utiliza um bit que armazena a informação do estado do objeto. Assim, esse bit passa a ser usado para verificar se o objeto deve ser atualizado pelo algoritmo. Como muitos outros algoritmos de consistência seqüencial, esse algoritmo é baseado na restrição de consistência WW (escrita-escrita), que serializa todas as operações de escrita da aplicação, não permitindo paralelismo nas operações de escrita. Além disso, esse algoritmo replica todos os dados compartilhados nos nós, permitindo que operações de leitura sejam executadas em paralelo.

No trabalho [9], Torres-Rojas propõe e apresenta um algoritmo de consistência para objetos compartilhados e distribuídos, chamado *Timed Consistency*. O algoritmo é baseado nos modelos de consistência seqüencial e causal. A principal diferença entre esse algoritmo e os outros é que ele utiliza o tempo do sistema para definir quando as operações de escrita devem estar disponíveis para as aplicações. No *Timed Consistency* os objetos compartilhados são replicados nos nós do aglomerado, permitindo que operações de leitura e escrita em objetos distintos possam ser executadas em paralelo. Com a replicação dos objetos, a sobrecarga de comunicação diminui nas aplicações que possuem muitas operações de leitura (não existe necessidade de requisitar o objeto), aumentando o desempenho do sistema.

Em [10], é proposto um *middleware* Java, chamado VSObjetcs (Virtual Shared Objects), para compartilhamento de objetos distribuídos em um aglomerado de computadores. O algoritmo de consistência desse *middleware* é baseado na restrição de consistência OO e não permite replicação dos objetos compartilhados. Nesse algoritmo, os objetos compartilhados migram entre os nós que os requisitam. Por esse motivo, não são permitidas operações paralelas sobre os mesmos objetos, porque só existe uma cópia do objeto compartilhado. A principal vantagem desse algoritmo, é que para alguns tipos de aplicação (sem acesso concorrente no mesmo objeto), o desempenho alcançado é superior aos outros algoritmos, pois, ele não fica atualizando ou invalidando as replicas dos objetos nos nós.

Observando os algoritmos de consistência que implementam modelos de consistência de memória, nós propomos e apresentamos em [19], um modelo estrutural que permite representar qualquer modelo de consistência uniforme ou híbrido. Esse modelo estrutural é um conjunto de módulos comportamentais que representam cada uma das etapas de um algoritmo de consistência (protocolo de coerência, restrições de consistência, protocolo de replicação, política de acesso e política de eventos).

Com o modelo estrutural, nós propomos, desenvolvemos e apresentamos em [1] [2] um modelo de consistência de objetos reconfigurável que foi representado e implementado através de um algoritmo reconfigurável de consistência, chamado RCA (*Reconfigurable Consistency Algorithm*). O RCA é um algoritmo que pode alterar seu comportamento (re-configuração) durante a execução de um conjunto de aplicações considerando as características da arquitetura e da carga de trabalho. Por exemplo, o RCA pode trocar a restrição de consistência, dependendo das necessidades do ambiente. Na reconfiguração, o RCA muda o seu comportamento para se tornar mais flexível e melhorar o desempenho das aplicações paralelas nos aglomerados de computadores [1] [2].

3. Trabalhos Relacionados

Nos últimos anos, muitos trabalhos e estudos de análise de desempenho de algoritmos de consistência têm sido realizados [11] [12] [13] [14] [15] [16] [17] [19]. Entretanto, a maioria desses utilizam modelos analíticos e não consideram algoritmos de consistência para *softwares* DSM baseados em objetos.

Entre os trabalhos encontrados, os artigos [12] [13] [14] [16] [18] estão mais relacionados ao nosso trabalho, pois fazem análise de desempenho de diferentes algoritmos de consistência, com simulação ou experimentos reais de cargas de trabalho reais.

No trabalho [12], os autores fazem uma análise de desempenho de dois algoritmos de consistência que representam o modelo de consistência seqüencial. Os dois algoritmos analisados foram propostos pelos autores que usaram simulação e cargas de trabalho sintéticas. O tempo total de simulação foi usado como métrica para análise de desempenho. Os resultados, obtidos na simulação, mostram que os algoritmos propostos pelos autores aumentam o desempenho do ambiente para a carga de trabalho selecionada em relação aos algoritmos propostos que utilizam primitivas de sincronização.

Em [13], Adve apresenta a análise de desempenho de algoritmos de consistência que representam modelos de consistência híbridos (Entrada e Liberação Preguiçosa). Eles utilizam aplicações reais, métrica tempo de resposta, um aglomerado de computadores composto de oito nós de processamento e dois *softwares* DSM para análise de desempenho dos diferentes algoritmos. Os resultados mostram que os algoritmos que implementam o modelo de consistência liberação preguiçosa apresentam ganhos de desempenho de até 41% em relação aos algoritmos que implementam o modelo de consistência entrada.

No trabalho [14], os autores apresentam um estudo do impacto de protocolos de coerência em diferentes algoritmos de consistência. O estudo foi realizado tanto para *softwares* DSM quanto para DSM implementados em *hardware*. A análise do impacto dos protocolos de coerência foi feita considerando apenas o modelo de consistência liberação preguiçosa. Modelos analíticos estruturados através de redes de Petri e autômatos foram usados. Os resultados mostram que nos DSM implementados em *hardware* o desempenho é maior quando relaxamos o protocolo de ordem de eventos. Por outro lado, nos *softwares* DSM, o desempenho é maior quando relaxamos o protocolo de coerência.

Zucker apresenta em [16] um estudo do desempenho de alguns modelos de consistência em arquiteturas multiprocessadas. Neste estudo, os autores usaram simulação, *speedup* como métrica de desempenho e uma arquitetura paralela com DSM implementado em hardware. Os modelos de consistência usados nesse estudo foram: seqüencial, fraca e liberação. Para os modelos de consistência seqüencial e fraca, os autores analisaram dois diferentes algoritmos de cada um. Os autores mostram, com seus resultados, que o desempenho de modelos de consistência relaxados é maior que os não relaxados, entretanto, esse ganho depende muito do computador paralelo e da carga de trabalho do ambiente.

No trabalho [18], os autores apresentam a análise de desempenho de diferentes protocolos de coerência em um mesmo modelo de consistência. Para análise de desempenho, os autores usaram um simulador desenvolvido por eles, quatro protocolos de coerência propostos na literatura e aplicações reais e sintéticas. A análise de desempenho foi realizada considerando os custos das operações, tolerância à falhas e escalabilidade de cada algoritmo. Os resultados apresentados no trabalho mostram que os protocolos de coerência de invalidação, em geral, possuem menor sobrecarga de comunicação que os protocolos de atualização. Entretanto, os protocolos de coerência de atualização são mais tolerantes à falhas do sistema.

Nos trabalhos [14] [18], os autores apresentam um estudo sobre o impacto dos diferentes tipos de protocolos de coerência nos modelos de consistência de memória uniforme e híbrido. Nos trabalhos [12] [13], é apresentada uma análise de desempenho de alguns algoritmos de consistência que representam os modelos de consistência seqüencial, entrada e de liberação preguiçosa. E, no trabalho [16] é apresentado um estudo completo da análise de desempenho de diferentes algoritmos de consistência para arquiteturas multiprocessadas. Entretanto, em nenhum desses trabalhos é apresentada uma análise de desempenho, com aplicações reais e sintéticas, de algoritmos que representam modelos de consistência de memória atômica, seqüencial e reconfigurável para *softwares* DSM baseados em objetos. Além disto, a maioria desses trabalhos avalia apenas, o impacto do protocolo de coerência e da política de acesso no desempenho.

4. Método Experimental

Nesta seção, nós apresentamos as métricas, o modelo computador paralelo e a carga de trabalho usada para análise de desempenho dos algoritmos de consistência. Além disso, nós apresentamos o nosso método experimental. O objetivo desses testes é varrer o maior espaço de tipos de aplicações e mostrar o impacto de cada etapa de diferentes algoritmos de

consistência no desempenho de aplicações sintéticas e reais em um aglomerado de computadores com *software* DSM baseado em objetos.

Para analisar o desempenho dos algoritmos de consistência, nós podemos usar diferentes métricas. A mais usada é o tempo de resposta [1] [2] [14]. A métrica tempo de resposta é definida na Eq. 1.

$$\text{TempoResposta} = \frac{\sum \text{TempoFinalTarefa} - \text{TempoSubmissãoTarefa}}{\text{NúmeroTarefas}} \quad \text{(Eq. 1)}$$

O computador paralelo usado nas nossas simulações é um aglomerado de computadores que executa um software DSM baseado em objetos. Na tabela 1, nós podemos observar as suas principais características. Seus valores foram obtidos a partir de benchmarks e bibliotecas de medição de desempenho (Sandra 2003, PAPI 2.3 etc.). Nós modelamos o nosso ambiente de testes na ferramenta de simulação ClusterSim [20] [21]. Essa ferramenta foi escolhida, pois permite a simulação de diferentes modelos de consistência e de softwares DSM baseados em objetos. Além disso, no ClusterSim é possível especificar características de uma aplicação (seu tipo, estrutura interna, numero de operações locais e número de operações remotas).

Tabela 1: Características da arquitetura

Característica	Valor
Número de nós	8 até 64
Freqüência do Processador	0.938 GHz
Ciclos por Instrução	0.9997105
Tamanho da Mensagem de Invalidação	96 bytes
Tamanho da Mensagem de Atualização	256 bytes – 4K
Rede	Fast Ethernet
Latência de Rede	0.000179 s
Tamanho Máximo de Segmento	1460
Largura de Banda (vazão medida)	11.0156 MBps
Sobrecarga de Protocolo	58 bytes

As aplicações que compartilham objetos podem ser divididas em três categorias: iterativas, *run-to-complete* e *fork and join* [22]. Neste trabalho, vamos analisar aplicações paralelas iterativas por possuírem um comportamento mais regular. Então, nós criamos duas cargas de trabalho. A primeira composta de quarenta aplicações sintéticas iterativas que foram agrupadas em quatro cargas sintéticas (10 aplicações para cada uma). Estas aplicações estão baseadas no nível de concorrência (nc) e em características importantes de aplicações paralelas desenvolvidas com o modelo de programação de variável compartilhada (número de objetos compartilhados, número de processos, padrões de acesso e número de operações de leitura e escrita). O nível de concorrência é baseado no número de objetos compartilhados para um conjunto de processos de uma aplicação paralela. Ele representa o nível de concorrência que uma aplicação terá durante a sua execução. O nível de concorrência pode ser mínimo, médio e máximo. No *nc* mínimo, existe pelo menos um objeto compartilhado para cada processo. No *nc* médio, existe no mínimo um objeto pra cada dois processos. E no *nc* máximo, o número de objetos compartilhados pelos processos em um mesmo instante é sempre um. A segunda carga de trabalho é composta de duas aplicações baseadas em aplicações reais (multiplicação de matrizes e *quicksort*).

A Tabela2, apresenta o número de aplicações de cada tipo de *nc* (mínimo, médio e máximo) para cada uma das quatro cargas sintéticas (CS) da primeira carga de trabalho. Além de serem baseadas no *nc*, essas aplicações também são baseadas no número de objetos compartilhados (1 até 8 objetos compartilhados), número de processos (1 até 8 processos), padrões de acesso (Leitura-Escrita, Escrita-Escrita, Leitura-Leitura e Escrita-Leitura), número de operações de leitura e escrita (10 a 80 operações). Essas aplicações foram executadas em um aglomerado de computadores com oito nós. Com estas aplicações, nós queremos observar o impacto do nível de concorrência nos algoritmos.

Tabela 2: Carga de trabalho sintética

Carga de Trabalho	nc máximo	nc médio	nc mínimo
Carga Sintética 1	4	4	2
Carga Sintética 2	4	2	4
Carga Sintética 3	2	4	4
Carga Sintética 4	0	8	2

A segunda carga de trabalho é composta de duas aplicações paralelas: a multiplicação de matrizes (3 objetos compartilhados que representam as matrizes) e o *quicksort* (1 objeto compartilhado que representa um vetor de 1000 inteiros). Com essas aplicações, nós queremos observar o desempenho dos algoritmos de consistência em aplicações reais de processamento intenso e a escalabilidade dos mesmos quando aumentamos o número de processos (8 até 64).

Para realizar a análise de desempenho dos algoritmos de consistência, nós comparamos os resultados obtidos com cada um dos cinco algoritmos para toda carga de trabalho simulada. Os algoritmos propostos em [1] [2] [5] [7] [8] [9] [10] já foram implementados no ClusterSim. Então, considerando a carga de trabalho e os cinco algoritmos de consistência nós realizamos 210 simulações (5 algoritmos x 42 aplicações).

5. Resultados

Nesta seção, nós apresentamos a análise de desempenho dos cinco algoritmos de consistência.

5.1 Carga de Trabalho Sintética

Nesta seção, nós apresentamos e analisamos os resultados obtidos para a primeira carga de trabalho (quatro cargas sintéticas). Para cada uma das quatro CS (carga sintética), nós apresentamos o tempo de resposta para todas as suas aplicações e algoritmos.

Na figura 1, nós apresentamos o tempo de resposta obtido com cada algoritmo de consistência para a CS 1. Como podemos observar, o RCA apresenta os melhores resultados para essa carga de trabalho. O RCA pode adaptar o seu comportamento ao tipo da arquitetura e da aplicação para obter o melhor desempenho. Por esse motivo, ele apresenta os melhores resultados para a CS 1.

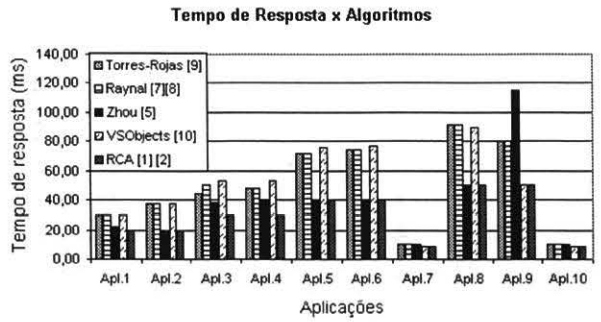

Figura 1: Tempo de Resposta para CS 1

Apesar do RCA ter obtido os melhores resultados, na maioria das aplicações da CS 1, os algoritmos Zhou [5] e VSObjetcs [10] apresentaram resultados semelhantes ao do RCA. Nas aplicações com nível de concorrência médio ou mínimo (7, 8, 9 e 10), o algoritmo proposto no VSObjetcs [10], obteve os mesmos resultados que o RCA. Esse algoritmo usa protocolo de replicação de migração, que é bom para aplicações onde existe pouca concorrência para acessar os objetos compartilhados (menor sobrecarga de comunicação). Nesse protocolo de replicação, cada nó possui o seu objeto e como a aplicação possui nível de concorrência mínimo as aplicações não ficam bloqueadas esperando por um objeto compartilhado. Por outro lado, nas aplicações com nível de concorrência máximo (2, 3, 4, 5 e 6), o algoritmo proposto em Zhou [5], obteve os mesmos resultados que o RCA. Esse algoritmo é melhor para aplicações com esse nível de concorrência, pois, permite que operações de leitura sejam executadas em paralelo, sequencializando apenas as operações de escrita.

O algoritmo proposto por Torres-Rojas [9] apresenta os piores resultados para essa carga de trabalho, figura 1. Esse resultado já era esperado, pois esse protocolo não permite que operações de leituras sejam executadas em paralelo. Assim, mesmo não existindo concorrência entre os processos da aplicação, esse algoritmo não aproveita essa característica para aumentar o número de operações em paralelo e conseqüentemente o seu desempenho. É importante observar que o algoritmo proposto por Raynal [8] apresenta resultados semelhantes ao algoritmo de Torres-Rojas para a maioria das aplicações. O algoritmo de Raynal torna-se melhor que o algoritmo de Torres-Rojas, para aplicações com nível de concorrência máximo. O algoritmo proposto por Torres-Rojas implementa o modelo de consistência atômico, que bloqueia um maior número de processos nesse caso. Esse modelo de consistência é mais forte que o algoritmo de Raynal.

Na figura 2, nós apresentamos o tempo de resposta obtido com cada algoritmo de consistência para a CS 2. Assim como na CS 1, o RCA apresenta os melhores resultados, sendo que, para algumas aplicações (1, 3, 4, e 5) ele apresenta os mesmos resultados que os algoritmos de Zhou [5] e VSObjects [10]. Na CS 2, o número de aplicações com nível de concorrência máximo é o mesmo de aplicações com nível de concorrência mínimo. Assim, o RCA pode manter constante o seu comportamento para essas aplicações.

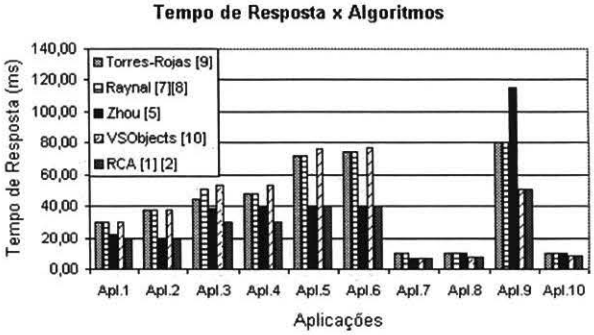

Figura 2: Tempo de Resposta para CS 2

Como podemos observar, o algoritmo proposto por Raynal [8], apresenta os piores resultados, na média, para as aplicações da CS 2. Esse algoritmo usa restrição de consistência OO e protocolo de coerência de invalidação. A restrição de consistência OO, não é uma boa opção para a CS 2, uma vez que, a maior parte das aplicações da CS 2 possui nível de concorrência

máxima. Essa restrição bloqueia um número muito grande de operações compartilhadas, pois não permite que operações no mesmo objeto sejam executadas ao mesmo tempo. Além disso, o protocolo de invalidação, invalida os objetos compartilhados, o que faz com que os processos necessitem requisitar novamente o objeto para a próxima operação.

È importante notar que a aplicação 9, apresentou os maiores tempos de respostas para todos algoritmos de consistência. Essa aplicação possui um grande número de objetos compartilhados. Então, independente do protocolo de replicação e de coerência utilizado, a sobrecarga de comunicação gerada pelo algoritmo para gerenciar as réplicas e os objetos é muito alta.

Na figura 3, nós apresentamos o tempo de resposta para as aplicações da CS 3. Entretanto, para a maioria das aplicações, o algoritmo VSObjects [10] obtém os mesmos resultados do RCA. Apenas na aplicação 1, o RCA se apresenta melhor que todos os outros algoritmos de consistência. Nessa aplicação, o nível de concorrência é máximo, e como o algoritmo de consistência do VSObjects utiliza protocolo de replicação de migração, os processos da aplicação ficam bloqueados esperando o objeto migrar.

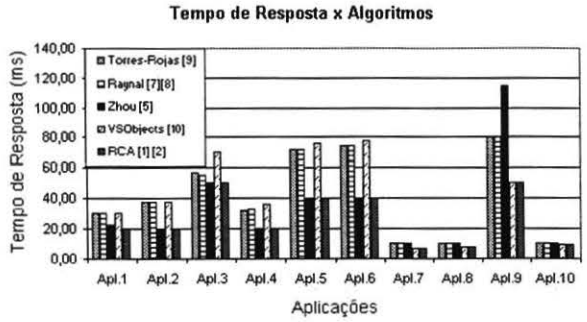

Figura 3: Tempo de Resposta para CS 3

Ainda na figura 3, podemos observar que o algoritmo proposto por Zhou [5] apresenta os piores tempos de resposta. Esse algoritmo utiliza restrição do tipo WW, que sequencializa as operações de escrita e impede que escritas concorrentes em objetos distintos sejam executadas em paralelo. Além disso, esse algoritmo utiliza protocolo de coerência de atualização, que para aplicações com nível de concorrência mínimo, gera grande sobrecarga de comunicação. Nesse tipo de aplicação não é necessário que os objetos sejam sempre atualizados, uma vez que, cada aplicação possui seu próprio objeto. Entretanto, o protocolo de atualização sempre atualiza os objetos com os novos valores. Como as mensagens de atualização dos objetos possuem o tamanho do objeto compartilhado, o tráfego na rede aumenta e conseqüentemente o tempo de resposta da aplicação.

Na figura 4, nós apresentamos os resultados obtidos para a CS 4. Como podemos observar, assim como nas demais cargas de trabalho sintéticas, o RCA obteve os melhores resultados. Entretanto, diferente, dos resultados obtidos com as outras cargas de trabalho, o RCA obteve resultados iguais para diferentes algoritmos de consistência para cada aplicação. Ou seja, para cada aplicação, além do RCA, sempre existe um outro algoritmo de consistência que também apresenta o melhor tempo de resposta.

A maioria das aplicações da CS 4, possui *nc* médio, e por esse motivo, o algoritmo proposto por Raynal [8], apresenta os piores resultados. Como já dissemos, esse algoritmo usa restrição de consistência OO e protocolo de coerência de invalidação. Essa restrição bloqueia um número muito grande de operações compartilhadas, pois não permite que operações no mesmo objeto sejam executadas ao mesmo tempo. Além disso, o protocolo de invalidação, invalida os objetos compartilhados, o que faz com que os processos necessitem requisitar novamente o objeto para a próxima operação. Além disso, é importante observar que os resultados obtidos com esse algoritmo são muito semelhantes aos resultados obtidos pelo algoritmo de Torres-Rojas. A principal diferença entre os dois algoritmos é observada em aplicações com *nc* médio, pois, o algoritmo de Raynal torna-se pior que o algoritmo de Torres-Rojas, mesmo sendo uma representação do modelo de consistência seqüencial que é mais fraco que o modelo de consistência atômico representado pelo algoritmo de Torres-Rojas.

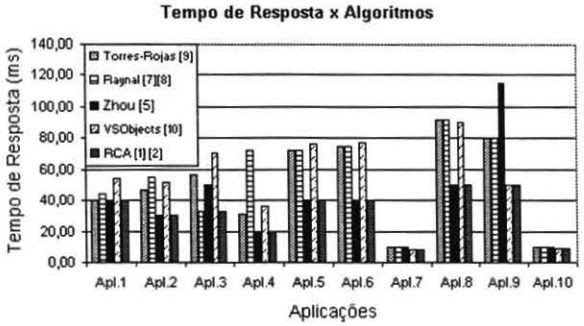

Figura 4: Tempo de Resposta para CS 4

5.2 Carga de Trabalho Real

Nesta seção, nós apresentamos os gráficos, em escala logarítmica, dos tempos de resposta da carga de trabalho real (multiplicação de matrizes e *quicksort*), obtidos para diferentes números de processos.

Na figura 5, podemos observar que o algoritmo de consistência RCA apresenta o melhor resultado para a aplicação de multiplicação de matrizes, para diferentes números de processos, pois ele reconfigura seu comportamento para obter o melhor resultado. Entretanto, para o número de processos igual a 16, o algoritmo de Zhou [5] apresenta o mesmo resultado. Nesse caso, o mecanismo de reconfiguração do RCA muda o comportamento dele para o mesmo do algoritmo de Zhou [5], dando ao RCA melhor desempenho. Nessa figura, podemos ainda observar, que mesmo variando o número de processos, o RCA continua apresentado os melhores resultados, pois ele pode se adaptar as mudanças no ambiente.

Os algoritmos de Torres-Rojas [9] e de Raynal [7] [8], apresentam na média, resultados semelhantes quando o número de processos é pequeno (até 16), figura 5. Nesse caso, como não existem muitos processos tentando acessar os mesmos objetos compartilhados, o algoritmo de consistência não tem tanto impacto no desempenho da aplicação. Entretanto, quando o número de processos aumenta muito (64 processos), o algoritmo de Raynal apresenta o menor tempo de resposta. Esse algoritmo possui política de eventos mais fraca (seqüencial) que a do algoritmo de Torres-Rojas (atômica).

Ainda na figura 5, podemos observar que o algoritmo VSObjects [10], apresenta os piores tempos de respostas para a aplicação de multiplicação de matrizes. Nesse algoritmo, o protocolo de replicação usado é o de migração, que não apresenta bom desempenho quando existe muito compartilhamento de dados, como é o caso da multiplicação de matrizes. Nessa aplicação cada matriz é representada por um objeto compartilhado, e movimentar esses objetos entre os nós gera muita comunicação.

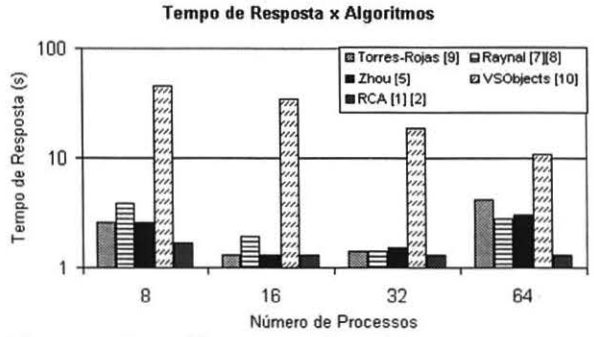

Figura 5: Tempo de Resposta para Multiplicação de Matrizes

Na figura 6, podemos observar que o algoritmo de Raynal apresenta resultados muito bons para a aplicação *quicksort* quando o número de processos é pequeno (16 processos). Esse algoritmo utiliza protocolo de coerência de invalidação, que para essa aplicação apresenta um bom desempenho e fez o tempo de resposta total diminuir. Nesse tipo de aplicação, as atualizações nos objetos compartilhados não precisam ser vistas por todos os nós em um mesmo instante, então, o protocolo de invalidação se torna uma boa opção já que o objeto só é atualizado, com o novo valor, na próxima operação de leitura.

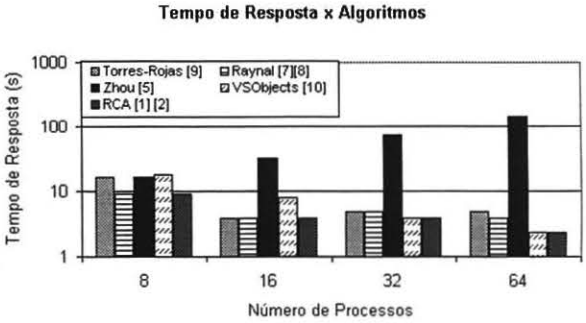

Figura 6: Tempo de Resposta para *Quicksort*

Podemos observar que o algoritmo Zhou [13], apresenta os piores tempos de respostas para a aplicação de quicksort (figura 6). Nesse algoritmo, o protocolo de coerência de atualização aumentou a sobrecarga de comunicação (a cada operação de escrita todas as réplicas eram atualizadas com o novo valor) e conseqüentemente o tempo de resposta da aplicação.

Os algoritmos de Torres-Rojas [9] e de Raynal [7] [8], apresentam, na média, resultados semelhantes para todos os números de processos. Para essa aplicação, esses algoritmos possuem comportamento semelhante, pois usam o mesmo tipo de protocolo de coerência e porque a aplicação possui apenas um objeto.

Ainda na figura 6, podemos observar que, assim como nas outras aplicações, o algoritmo de consistência RCA apresenta os melhores resultados, para diferentes números de processos, pois ele se reconfigura para obter o melhor resultado.

6. Conclusões

Neste artigo, nós analisamos o desempenho de cinco algoritmos de consistência [1] [5] [8] [9] [10] que implementam os modelos de consistência seqüencial e atômico, através de simulação, em um aglomerado de computadores com *software* DSM baseado em objetos.

Em relação à análise de desempenho, nós verificamos que o algoritmo RCA, proposto em [1] [2], apresentou os melhores resultados. Como apresentado na seção 5, esse resultado já era esperado, uma vez que o RCA possui um mecanismo de reconfiguração que

permite adaptar o seu comportamento para cada tipo de arquitetura e carga de trabalho. O algoritmo proposto por Raynal [8], apresentou os piores resultados para cargas de trabalho sintéticas com nível de concorrência era máximo e um dos melhores resultados para aplicações reais. Os algoritmos de consistência Zhou [5] e VSObjects [10] apresentaram os piores resultados para aplicações reais, pois utilizam protocolo de coerência de atualização e protocolo de replicação de migração. No entanto, para as aplicações sintéticas eles apresentaram um dos melhores resultados. O algoritmo Torres-Rojas [9], apesar de representar um modelo de consistência atômico, apresentou resultados muito bons quando comparado com os outros algoritmos.

Neste trabalho, nós verificamos que o protocolo de coerência de um algoritmo de consistência é responsável pelo maior impacto no desempenho das aplicações em arquiteturas paralela com *software* DSM. Os protocolos de coerência são responsáveis pela maior parte da sobrecarga de comunicação gerada pelos algoritmos de consistência. Além disso, nós verificamos que um algoritmo de consistência que representa o modelo de consistência atômico (mais forte) pode gerar resultados melhores que alguns algoritmos que representam o modelo de consistência seqüencial dependendo da carga de trabalho.

A principal contribuição desse trabalho é a análise de desempenho dos algoritmos de consistência e do impacto de cada uma das suas etapas no desempenho. Como trabalhos futuros nós destacamos: proposta de algoritmos de consistência que levem em consideração o nível de concorrência das aplicações, implementação e testes em um ambiente real.

7. Referências

[1] C. V. Pousa, L. F. W. Goes, C. A. P. S. Martins, "Reconfigurable Object Consistency Model". *7th Workshop on Advances in Parallel and Distributed Computational Models*, IPDPS, 2005.

[2] C. V. Pousa, D. O. Penha, L. F. W. Goes, C. A. P. S. Martins,"Reconfigurable Sequential Consistency Algorithm". *12th Reconfigurable Architectures Workshop*, IPDPS, 2005.

[3] A. C. M. A. Melo, "Defing Uniform and Hybrid Memory Consistency Models on a Unified Framework". 32th HICSS, Vol VIII-Software Technology, 1999, pp. 270-279.

[4] M. Mizuno, M. Raynal, J. Z. Zhou, "Sequential Consistency in Distributed Systems: Theory and Implementation". *Technical Report RR-2347, INRIA*, 1995.

[5] J. Z. Zhou, M. Mizuno, G. Singh, "A Sequentially Consistent Distributed Shared Memory". *Int. Conference on Computing and Information*, 1993, p.165-169.

[6] D. Wang, I. Chen, and C. Chu, "Analyzing reconfigurable algorithms for managing replicated data with strict consistency requirements: a case study". *24th Annual International Computer Software and Applications Conference*, 2000, pp.608 – 613.

[7] M. Raynal, K. Vidyasankar, "A Distributed Implementation of Sequential Consistency with Multi-Object Operations". *24th IEEE Int. Conf. on Distributed Computing Systems*, IEEE Computer Society Press, 2004, pp. 544-551.

[8] M. Raynal, "Sequential Consistency as Lazy Linearizability". *14th ACM Symposium on Parallel Algorithms and Architectures*, 2002, pp. 151-152.

[9] F. J. Torres-Rojas, M. Ahamad, M. Raynal, "Timed Consistency for Shared distributed Objects". *Symposium on Principles of Distributed Computing*, 1999, pp 163-172.

[10] C. V. Pousa, D. O. Penha, C. A. P. S. Martins, "VSObjects: Middleware para Gerenciamento de Objetos Virtualmente Compartilhados". *Workshop em Sistemas Computacionais de Alto Desempenho*, 2003.

[11] C. Amza, A.L. Cox, S. Dwarkadas, L.-J. Jin, K. Rajamani, and W. Zwaenepoel, "Adaptive Protocols for Software Distributed Shared Memory". *IEEE, Special Issue on Distributed Shared Memory*, Vol. 87, 1999, pp. 467-475.

[12] G. Girad, H. F. Li, "Evaluation of Two Optimized Protocols for Sequential Consistency", *Thirty-second Annual Hawaii International Conference on System Sciences*- Volume 8, 1999, pp. 8009.

[13] S. V. Adve, A. L. Cox, S. Dwarkadas, R. Rajamony, W. Zwaenepoel, "A Comparison of Entry Consistency and Lazy Release Consistency Implementations" 2nd *IEEE Symp. on High-Performance Computer Architecture* , 1996.

[14] W. Shi, W. Hu, Z. Tang, "An interaction of coherence protocols and memory consistency models in DSM systems", *ACM SIGOPS Operating Systems Review*, Volume 31, Issue 4, 1997, pp. 41 – 54.

[15] M. Stumm and S. Zhou, "Algorithms Implementing Distributed Shared Memory", *IEEE Computer*, vol. 23, no. 5, 1990, pp. 54-64.

[16] R. N. Zucker, J.-L. Baer, "A Performance Study of Memory Consistency Models". *19th International Symposium on Computer Architecture*, 1992.

[17] K. Gharachorloo, A. Gupta, J. Hennessy, "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors", *SIGPLAN Notices*, 1991.

[18] S. K. Shah and B. D. Fleisch, "A Comparison of DSM Coherence Protocols using Program Driven Simulations". *International Conference on Parallel and Distributed Processing Techniques and Applications*, 1997.

[19] C. V. Pousa e C. A. P. S. Martins, "Modelo Estrutural para Modelos de Consistência", Relatório Técnico PPGEE, Puc Minas, 2005.

[20] L. F. Góes, L. E. S. Ramos, C. A. P. S. Martins, "ClusterSim: A Java-Based Parallel Discrete-Event Simulation Tool for Cluster Computing", *IEEE International Conference on Cluster Computing*, 2004.

[21] C. V. Pousa, L. E. Ramos, L. F. W. Góes, C. A. P. S. Martins, "Extending ClusterSim With MP And DSM Modules", *Workshop High Performance Computer Science and Engineering*, 2004.

[22] Y.T. Liu, T.Y. Liang, Z. H. Kuo, C. K. Shich, "Involving Memory Resource Consideration into Workload Distribution for Software DSM Systems", *International Workshop on Distributed Shared Memory*, CCGrid, 2004.