Desafios de Consistência e Progresso em Memória Transacional em Software

Cátia Mesquita Brasil Khouri[1], Fabíola Gonçalves Pereira Greve[2]

[1] Departamento de Ciências Exatas – Universidade Estadual do Sudoeste da Bahia (UESB) – Vitória da Conquista – BA. Aluna do DMCC UFBA/UEFS/UNIFACS

[2] Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)

catia091@dcc.ufba.br, fabiola@dcc.ufba.br

Resumo. *A crescente utilização de sistemas multiprocessadores tem incentivado fortemente a busca por novas metodologias, linguagens e ferramentas que facilitem o desenvolvimento de programas concorrentes corretos e eficientes. Alternativas às técnicas baseadas em bloqueios têm sido estudadas a exemplo de memória transacional, podendo ser implementada em hardware ou software. Entretanto, muitos aspectos relativos à semântica de memória transacional em software precisam ser melhor investigados.*

1. Introdução

A construção de programas e estruturas de dados concorrentes é um problema que vem sendo objeto de estudo ao longo dos anos. Mais recentemente, a utilização de computadores *multicore*, tem aumentado o interesse no tema. Para que se possa tirar o máximo proveito de arquiteturas paralelas, é preciso que o software seja explicitamente paralelizado. Atualmente, o modelo de programação concorrente mais utilizado é o *multithreaded*, em que *threads* concorrentes comunicam-se através de escritas e leituras em localizações de memória compartilhadas. Os mecanismos de sincronização tradicionalmente utilizados baseiam-se no uso explícito de bloqueios, que muitas vezes apresentam problemas como dificuldade de composição, degeneração do paralelismo e desempenho e dificuldade de implementação. Erros na colocação de bloqueios podem causar instabilidades no sistema como impasse (*deadlock*) e inversão de prioridade.

Baseada no conceito de transação, amplamente utilizado no âmbito dos bancos de dados, memória transacional tem sido proposta como mecanismo de sincronização em sistemas *multithreaded*. **Memória transacional** provê construtores de programação que permitem especificar blocos de programa que devem ser executados atomicamente e se encarrega de que o acesso a dados compartilhados seja realizado consistentemente.

2. Memória Transacional

Memória transacional (*TM – Transactional Memory*) [Herlihy, M.; Moss, J. E., 1993] é um mecanismo que toma emprestadas as noções de atomicidade, consistência e isolamento das transações de bancos de dados (*DB – data base*) [Silberschatz; Korth; Sudarshan, 2005]. Permite aos programadores definir operações customizadas de leitura e escrita que se aplicam a palavras de memória independentemente escolhidas. Para

implementar TM, Herlihy e Moss propõem modificações a arquitetura convencional, acrescentando uma cache primária, alterando o protocolo de coerência de cache e acrescentando algumas instruções ao processador. A partir dessa proposta, muitas aplicações de TM baseadas em hardware (HTM) foram apresentadas na literatura [Larus e Rajwar, 2006].

Shavit, M. e Touitou, D. (1996) introduziram a idéia de *Software Transactional Memory* (STM), um projeto que suporta programação transacional de operações de sincronização em software de modo flexível. Uma STM é um objeto compartilhado que se comporta como uma memória, a qual suporta múltiplas alterações em seus endereços por meio de transações – sequências finitas de operações primitivas sobre a memória. STM permite ao programador escrever uma sequência de instruções (uma transação) sem se preocupar com a associação entre bloqueio e dados e granularidade de bloqueios, por exemplo. Apesar dos avanços proporcionados por STM, diversos desafios ainda restam no intuito de permitir a utilização prática da abstração [Feber, P. et al. (2008)].

3. Desafios em Memória Transacional em Software

A despeito das semelhanças, transações de memória diferem de transações de DB em diversos aspectos importantes, os quais se constituem em interessantes oportunidades de pesquisa. Ao contrário do que ocorre com DB, a semântica de TM ainda não está bem definida formalmente e questões relacionadas a consistência e garantias de progresso oferecidas pelas transações precisam ser investigadas. Feber, P. et al. (2008) discutem algumas dessas questões, apresentadas abaixo, que são o foco deste trabalho.

3.1 Isolamento de Transações

Para garantir a consistência, é fundamental assegurar o isolamento das transações. No âmbito de DB, a condição de corretude básica é que o resultado da execução de determinadas transações concorrentes deve ser o mesmo que seria obtido se elas fossem executadas sequencialmente (*serializability* – capacidade de serialização). Isso, porém, não define o que acontece a uma transação abortada. Na realidade, nada impede que uma transação abortada observe um estado inconsistente do sistema, o que pode causar vários problemas. Ela pode levantar exceções inesperadas, entrar em *laços* infinitos ou acessar endereços de memória inválidos. Em sistemas de DB o acesso a dados compartilhados é controlado, pois todas as operações ocorrem em transações. Já em TM, o controle é mais difícil, pois dados são acessados diretamente e operações não transacionais ocorrem simultaneamente. Então, a semântica dessa interação entre operações transacionais e não transacionais precisa ser completamente definida.

3.2 Transformação de Código Concorrente Legado

Uma opção na transformação de código escrito sem suporte transacional, é encapsular todas as operações não-transacionais em uma transação que não possa abortar. Isso, porém, não é trivial. Outra possibilidade é só exigir consistência do código transacional; é como se no programa houvesse um único bloqueio envolvendo alguns *trhreads* enquanto que outros não precisariam adquirir este bloqueio. Assim, *threads* acessando

objetos fora de transações não possuem garantia de consistência. Obviamente, é preciso estudar com cuidado o impacto disso sobre a semântica das transações.

3.3 Medidas de Desempenho de Sistemas Transacionais

O custo de armazenar dados em disco é muitas ordens de magnitude superior ao custo de armazenar numa memória compartilhada. Estratégias de otimização para sistemas de DB são, portanto, diferentes daquelas para TM. Se, por um lado, serializar o acesso ao disco melhora o desempenho no primeiro caso, em TM é mais indicado o acesso concorrente. O objetivo é aumentar o *throughput* – manter os processadores trabalhando e executar o maior número possível de transações que permaneçam vivas e culminem com a validação. Alguns estudos começam a ser desenvolvidos, mas ainda há muito a ser feito no sentido de garantir a sobrevivência e o progresso (*liveness*) de transações.

3.4 Estado de uma Transação

Quando uma transação aborta, suas atualizações de dados devem ser anuladas. Desta maneira, um sistema TM deve ser capaz de checar os dados compartilhados pelas transações. Diferente das tabelas de DB, as quais possuem um formato bem definido, os dados compartilhados em TM podem ter os diferentes tipos permitidos pela linguagem de programação. Além disso, o sistema pode prever atualização direta – fazendo cópias dos dados originais para o caso deles terem que ser copiados de volta na a memória; ou atrasada – as alterações são feitas em variáveis locais e transferidas para a memória quando a transação é validada. Estas decisões de projeto afetam o desempenho da implementação e a complexidade do gerenciamento de estado. No momento de uma leitura ou escrita, o sistema precisa decidir para qual versão do dado redirecionar o acesso (versão antiga, local ou o dado real). E mais, é desejável poder distinguir entre acessos de leitura e escrita, pois o primeiro é menos conflitante que o segundo. Estas são outras questões que devem ser investigadas.

4. Conclusão

Memória transacional é um campo de pesquisa promissor e que apresenta muitos desafios e perspectivas para o desenvolvimento de sistemas concorrentes. Espera-se, com a conclusão deste trabalho de tese, contribuir para a resolução de alguns dos desafios anteriormente apresentados.

Referências

Feber, P. et al. (2008) “Transactions are back—but are they the same?” Em ACM SIGACT News, vol. 39, nº. 1. Março.

Herlihy, M.; Moss, J. E. (1993) “Transactional Memory: Architectural Support for Lock-Free Data Structures”. Em Proceedings of the 20th Annual International Symposium on Computer Architecture, pp. 289–300. Junho.

Larus, J. R. e Rajwar, R. (2006). “Transactional Memory”. Morgan & Claypool Publishers, USA.

Shavit, M. e Touitou, D. (1996) “Software Transactional Memory”. Em Distributed Computing, pp. 99-116. Agosto. Srpringer.

Silberschatz, A.; Korth, H. F.; Sudarshan, S. (2005). “Database System Concepts”. McGraw Hill, USA.