Avaliação do novo sistema de execução de transações para CMTJava [†]

Rafael de Leão Bandeira[1], **André Rauber Du Bois**[1], **Maurício Lima Pilla**[1]

[1]Programa de Pós-Graduação em Computação
Centro de Desenvolvimento Tecnológico – UFPEL

{rdlbandeira, dubois, pilla}@inf.ufpel.edu.br

Resumo. *Neste artigo descreve-se a implementação de um novo sistema de execução de transações na CMTJava. São apresentadas as principais características desse algoritmo e também resultados preliminares do desempenho obtido com o novo sistema.*

1. Introdução

Memória transacional (MT) é um mecanismo de controle de concorrência no acesso à memória compartilhada. Esse método é baseado nas transações de bancos de dados e visa tornar a escrita de programas concorrentes mais simples. CMTJava [Du Bois e Echevarria 2009] é uma extensão de Java para programação com MT. Na linguagem, o programador realiza o acesso à memória compartilhada dentro de transações atômicas. Se alguma operação falhar, toda a transação é executada novamente.

Todo modelo de MT conta com um sistema transacional, responsável por executar as transações, detectando e resolvendo conflitos. Neste artigo descreve-se a implementação de um novo sistema de execução de transações na CMTJava. São apresentadas as principais características desse algoritmo e também resultados preliminares da desempenho obtido com o novo sistema.

2. Sistema Transacional

O novo sistema transacional desenvolvido baseia-se no algoritmo descrito em [Dragojević et al. 2009]. A detecção de conflitos entre transações de escrita é feita de forma adiantada. Como nesse caso tipicamente é necessário abortar uma das transações, evita-se assim que a transação conflitante seja executada até o fim, para só então ser abortada. Já os conflitos entre uma transação de leitura e outra de escrita são detectados tardiamente. Isto permite maior paralelismo na execução, uma vez que esse tipo de conflito pode muitas vezes ser resolvido sem cancelamentos. No sistema transacional anterior, os conflitos são sempre detectados de forma tardia.

Cada atributo de um objeto acessado transacionalmente é provido de dois *locks* versionados: um *lock* de leitura, também utilizado para guardar o indicador de versão do atributo, e um *lock* de escrita. Quando uma transação T realiza uma operação de escrita em um atributo, esta tenta adquirir o *lock* de escrita associado ao mesmo. Se esta

[†]Este trabalho foi financiado pelos projetos FAPERGS/PRONEX/CNPq GREEN-GRID e FAPERGS/PESQUISADOR GAÚCHO.

aquisição falhar então outra transação encontra-se alterando tal atributo e T é abortada. Nas operações de leitura nenhum *lock* é adquirido. Ocorre apenas uma verificação de que o *lock* de leitura do atributo encontra-se livre.

Para efetivar as computações na memória, a transação adquire os *locks* de leitura de todos atributos alterados. Isto serve para que nenhuma operação de leitura nos atributos do conjunto de escrita seja realizada por uma transação concorrente. Os *locks* de escrita já encontram-se de posse da transação, pois são adquiridos a cada operação de escrita.

3. Resultados

Para avaliar o desempenho do sistema transacional, realizou-se um experimento com árvore rubro-negra. Os testes foram executados utilizando um processador de 4 núcleos com *HyperThreading*. Foram efetuadas operações de travessia na árvore (busca por elementos) e de atualizacão (inserção e remoção de elementos), variando o número de *threads*. A Figura 1 mostra os resultados obtidos utilizando o novo sistema transacional desenvolvido e a implementação anterior, com 20% de operações de atualização.

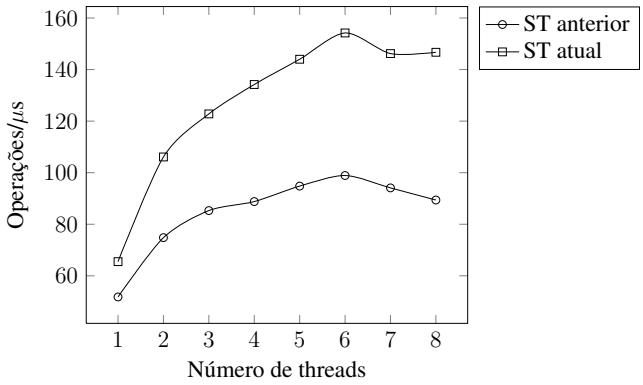

Figura 1. Comparação do número de operações realizadas em árvore rubro-negras com 20% de operações de atualização

4. Conclusões

O sistema transacional desenvolvido consegue realizar mais operações por unidade de tempo. Isso se deve ao fato do algoritmo utilizar uma política mais eficiente para detecção de conflitos, que evita a execução de transações condenadas a abortar.

Referências

Dragojević, A., Guerraoui, R., e Kapalka, M. (2009). Stretching transactional memory. *Sigplan Notices*, 44.

Du Bois, A. R. e Echevarria, M. (2009). A domain specific language for composable memory transactions in java. In *DSL '09: Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages*, pages 170–186, Berlin, Heidelberg. Springer-Verlag.