ESTUDO SOBRE STM PARA ARQUITETURAS DISTRIBUÍDAS

JERÔNIMO DA CUNHA RAMOS; MAURÍCIO LIMA PILLA; ANDRÉ RAUBER DU BOIS

Universidade Federal de Pelotas (UFPel) – {jdcramos,dubois,pilla}@inf.ufpel.edu.br

1. INTRODUÇÃO

Para tirar proveito do paralelismo disponibilizado pelos computadores atuais, os programas devem conter atividades que possam executar concorrentemente. Normalmente, em arquiteturas de memória compartilhada, isto é feito com o uso de threads, variáveis compartilhadas e *locks*. Porém, este modelo de programação impõe quase toda a complexidade de exploração do paralelismo e de tratamento das condições de corrida ao programador, o que torna-o propenso a erros, como *deadlocks*. A comunidade científica vem propondo alternativas a este modelo para facilitar e popularizar a programação concorrente, como é o caso das memórias transacionais (RIGO; CENTODUCATTE; BALDASSIN, 2007) .

Memória transacional é uma abstração para programação concorrente baseada na ideia de transações, similares às de banco de dados. Uma transação de memória é uma sequência atômica de operações que modificam a memória e que podem ser executadas completamente ou podem ser abortadas. Desta maneira, transações de memória executam como se estivessem modificando uma área de memória isolada do acesso de outras transações. Estas só conseguem ver o resultado após o *commit* (DU BOIS, 2008). Este modelo possui várias vantagens sobre o modelo clássico de programação, sendo as principais: ausência de *deadlock*, possibilidade de abortar e retornar ao estado original do programa, maior possibilidade de exploração do paralelismo, facilidade de programação, escalabilidade e composabilidade (RIGO; CENTODUCATTE; BALDASSIN, 2007; DU BOIS, 2008).

Nos últimos anos as pesquisas sobre Memórias Transacionais têm sido bastante focadas em máquinas *multicore*, deixando outros tipos de arquiteturas, como *clusters* e máquinas NUMA (*Non-Uniform Memory Access*) quase inexploradas (LU; WANG; LU, 2010; KOTSELIDIS et al., 2008; BOCCHINO; ADVE; CHAMBERLAIN, 2008). Estas arquiteturas tem como principal diferencial o tempo variado de acesso à memória local e remota. No caso dos clusters e outros Sistemas Distribuídos (SD), cada nodo possui seu próprio espaço de endereçamento de memória, fazendo com que a comunicação tenha que ser feita por troca de mensagens e, geralmente, não haja coerência de *cache*.

O modelo de memórias transacionais tem se mostrado bastante promissor e largamente estudado para aplicação em máquinas *multicore*. Este modelo também pode ser utilizado, com algumas adaptações, para facilitar a programação de arquiteturas distribuídas, como foi demostrado em RAMOS; DU BOIS; PILLA (2016), onde foi proposta uma linguagem de programação com suporte a transações envolvendo tanto objetos locais quanto distribuídos.

O estudo das arquiteturas distribuídas continua extremamente importante nos dias de hoje, tanto por comporem uma grande parcela da computação de alto desempenho, quanto pela tendência de cada vez mais dispositivos computacionais se comunicarem entre si. Além disso, com o aumento do uso de *clouds* computacionais, se torna importante o estudo de como sincronizar as ações que têm donos diferentes, utilizando as primitivas de memórias

transacionais. Sendo assim, a principal motivação deste trabalho é buscar meios para utilização de mecanismos de memórias transacionais em sistemas distribuídos com características diversas, avaliando o desempenho destes.

2. METODOLOGIA

Este trabalho está em etapa inicial e propõe um estudo das necessidades especiais para o uso de memórias transacionais em sistemas distribuídos, com objetivo central de desenvolver uma arquitetura para transações que envolvam objetos distribuídos. Para tal, a investigação será dividida em dois níveis: nível de linguagem e nível de execução.

No **nível de linguagem** serão investigadas como as primitivas de alto nível existentes na literatura para expressar transações de memória de maneira composicional (e.g. "*retry*" e "*or else*" (HARRIS et al., 2005)) se adaptariam em um contexto de transações distribuídas, além de primitivas adicionais que se mostrem necessárias.

No **nível de execução** será estudado como devem ser adaptados os algoritmo para transações concorrentes existentes, de forma a agregar o suporte à um contexto distribuído. Além disso, devem ser avaliadas as soluções existentes para ambientes distribuídos, identificando formas de como elas poderiam ser adaptadas para solucionar os problemas a nível de linguagem de programação. Também devem ser avaliadas técnicas para minimizar o impacto da latência de rede na comunicação.

Até o momento, foi realizada uma revisão do estado da arte da área. Existem diversas implementações de sistemas de memórias transacionais para máquinas *multicore*, tais como TinySTM (FELBER; FETZER; RIEGEL, 2008) , SwissTM (DRAGOJEVIĆ ; GUERRAOUI; KAPALKA, 2009), AdaptSTM (PAYER; GROSS, 2011) e CMTJava (DU BOIS; ECHEVARRIA, 2009), porém o foco destas não está em arquiteturas distribuídas, sendo assim não preveem soluções para os dificultantes citados acima. As propostas focadas em arquiteturas distribuídas são pouco numerosas, sendo estas focadas em *clusters*, como é o caso de Cluster-STM (BOCCHINO; ADVE; CHAMBERLAIN, 2008), DiSTM (KOTSELIDIS et al., 2008) e TFA (SAAD; RAVINDRAN, 2012).

Atualmente, está sendo realizada uma análise que aborda as necessidades especiais para o uso de memórias transacionais em sistemas distribuídos, estudando também as estratégias utilizadas nos trabalhos relacionados. Também estão sendo estudadas as primitivas de alto nível existentes na literatura para expressar transações de memória de maneira composicional. Esta etapa visa definir como e quais as extensões deverão ser implementadas em um contexto distribuído.

Em paralelo, estão sendo definidas as principais classes e interfaces de programação de nível intermediário. Esta etapa visa modularizar a arquitetura do sistema transacional para que este suporte a implementação dos diversos algoritmos abordados na bibliografia, bem como, é importante para tornar viável a realização de testes do sistema transacional sem a dependência da linguagem de domínio específico de alto nível.

A segunda parte da revisão bibliográfica visa investigar na literatura quais os algoritmos transacionais melhor se adaptam para dar suporte as primitivas transacionais em um contexto distribuído. Para validar a solução investigada, será desenvolvido um protótipo e um conjunto de *benchmarks* de teste. Nossa solução também será validada através do envio de artigos com as principais ideias propostas para conferências de importância na área.

3. RESULTADOS E DISCUSSÃO

Até o momento foram realizadas grande parte da revisão bibliográfica e parte da modelagem do sistema transacional distribuído, bem como, foi iniciada a etapa de implementação deste. Na revisão bibliográfica, foram estudados os principais trabalhos da área, avaliando as estratégias utilizadas tanto em nível de linguagem quanto em nível de sistema de execução local e distribuído. Já foram definidas as principais interfaces de nível intermediário, que servirão para "interligar" o nível de linguagem ao nível de execução. As interfaces principais interfaces já propostas podem ser vistas na Figura 1.

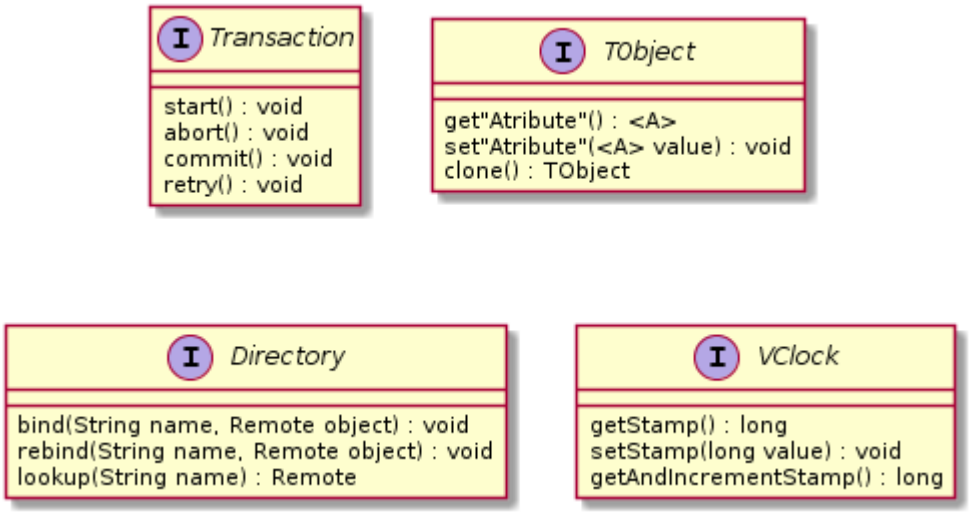

Figura 1 – Interfaces de Nível Intermediário para o Sistema Transacional Distribuído

A interface *Transaction* contém os métodos responsáveis por inicializar os metadados de uma transação, voltar ao estado anterior e fazer as validações necessárias para efetivar uma transação. A interface *TObject* contém os *getters* e os *setters* de cada atributo de um objeto transacional; estes métodos podem ser gerados sinteticamente pelo compilador e devem conter todas as ações necessárias para leitura e escrita dos atributos. *TObject* também pode ser clonado, para o caso de migração do objeto transacional entre nodos. A interface *Directory* é responsável pelo registro e localização dos objetos transacionais distribuídos através dos nodos. Já a interface *VClock* contém os métodos necessários para utilizar um relógio global virtual, que é utilizado para realizar o versionamento dos dados.

Uma versão do sistema transacional já está sendo implementada, utilizando o algoritmo TFA e, assim que pronta e testada, será submetida a *benchmarks* clássicos para transações.

4. CONCLUSÕES

Com a implementação da interface de nível intermediário proposta, será possível escrever programas utilizando-a, sem depender da conclusão da implementação do nível de linguagem. Isto facilitará os testes do nível de execução e o torna mais modular.

Espera-se que no final desta pesquisa obtenha-se uma solução de alto nível para o problema de transações utilizando objetos distribuídos, dando suporte

tanto a nível de linguagem, com abstrações de alto nível para a especificação de transações, quanto a nível de sistema de tempo de execução, provendo execução eficiente dessas abstrações. Neste sentido, para validar a solução investigada será desenvolvido um protótipo e *benchmarks* de teste.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BOCCHINO, R. L.; ADVE, V. S.; CHAMBERLAIN, B. L. Software Transactional Memory for Large Scale Clusters. **Proceedings of the 13th ACM SIGPLAN Symposium on PPoPP,** New York, p.247-258, 2008.

DRAGOJEVIĆ, A.; GUERRAOUI, R.; KAPALKA, M. Stretching Transactional Memory. **Proceedings of ACM SIGPLAN Conference on PLDI,** New York, p.155-165, 2009.

DU BOIS, A. R. Memórias Transacionais e Troca de Mensagens: Duas Alternativas para a Programação de Máquinas Multi-Core. **SBC, P. A. (Ed.). Escola Regional de Alto Desempenho,** p.43-76, 2008.

DU BOIS, A. R.; ECHEVARRIA, M. A. Domain Specific Language for Composable Memory Transactions in Java. **DSL '09: Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages,** p.170-186, 2009.

FELBER, P.; FETZER, C.; RIEGEL, T. Dynamic Performance Tuning of Word-Based Software Transactional Memory. **Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming**, New York, NY, USA, p.237-246, 2008.

KOTSELIDIS, C.; ANSARI, M.; JARVIS, K.; LUJÁN, M.; KIRKHAM, C.; WATSON, I. DiSTM: A Software Transactional Memory Framework for Clusters. **Proceedings of 37th ICPP,** Washington, p.51-58, 2008.

LU, K.; WANG, R.; Lu, X. Brief Announcement: NUMA-Aware Transactional Memory. **Proceedings of the 29th ACM SIGACT-SIGOPS**, New York, NY, USA, p.69-70, 2010.

PAYER, M.; GROSS, T. R. Performance Evaluation of Adaptivity in Software Transactional Memory. **IEEE International Symposium on Performance Analysis of Systems and Software**, p.165-174, 2011.

RAMOS, J. C.; DU BOIS, A. R.; PILLA, M. L. An Embedded Domain Specific Language for Distributed Memory Transactions in Java. **Proceedings of the 31st Annual ACM Symposium on Applied Computing**, New York, NY, USA, p.1980-1985, 2016.

RIGO, S.; CENTODUCATTE, P.; BALDASSIN, A. Memórias Transacionais: Uma Nova Alternativa para Programação Concorrente. **Minicursos do VIII WSCAD,** 2007.

SAAD, M. M.; RAVINDRAN, B. Transactional Forwarding: Supporting Highly-Concurrent STM in Asynchronous Distributed Systems. **SBAC-PAD, 2012 IEEE 24th International Symposium**, p.219-226, 2012.